Alignment-free sequence comparison using maximal common substrings

Burkhard Morgenstern

Dept. Bioinformatics
Institute of Microbiology and Genetics (IMG)
University of Göttingen

14. December 2017

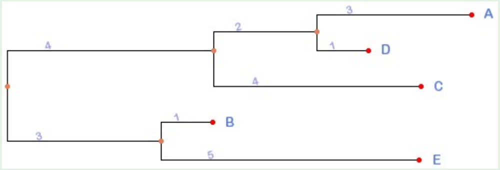
Spaced Words - recap

Spaced Words - recap

- *Input:* pairwise distances between ‘objects’
- *Output:* tree, with ‘objects’ at tips, representing distances.

Example (Distance matrix and tree representing distances)

	A	*B*	*C*	*D*	*E*
A	0	13	9	4	17
B		0	12	11	6
C			0	7	16
D				0	15
E					0



Spaced Words - recap

Simplest approach:

Spaced Words - recap

Simplest approach:

- Define *distances* between two sequences as (estimated) number of substitutions per position

Spaced Words - recap

Simplest approach:

- Define *distances* between two sequences as (estimated) number of substitutions per position
- Estimate number of substitutions per position based on number of mismatches in alignments with *Jukes-Cantor*

Spaced Words - recap

Simplest approach:

- Define *distances* between two sequences as (estimated) number of substitutions per position
- Estimate number of substitutions per position based on number of mismatches in alignments with *Jukes-Cantor*

Example (Pairwise sequence alignment)

S_1	T	C	A	C	G	T	C	G	T	C	G	
S_2	A	C	A	T	C	G	A	G	C	G	A	G

Spaced Words - recap

Simplest approach:

- Define *distances* between two sequences as (estimated) number of substitutions per position
- Estimate number of substitutions per position based on number of mismatches in alignments with *Jukes-Cantor*

Example (Pairwise sequence alignment)

S_1	T	C	A	C	G	T	C	G	$-$	T	C	G	$-$	$-$
S_2	$-$	$-$	A	C	A	T	C	G	A	G	C	G	A	G

Spaced Words - recap

Simplest approach:

- Define *distances* between two sequences as (estimated) number of substitutions per position
- Estimate number of substitutions per position based on number of mismatches in alignments with *Jukes-Cantor*

Example (Pairwise sequence alignment)

S_1	*T*	*C*	*A*	*C*	*G*	*T*	*C*	*G*	–	*T*	*C*	*G*	–	–
S_2	–	–	*A*	*C*	*A*	*T*	*C*	*G*	*A*	*G*	*C*	*G*	*A*	*G*

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{llllllll} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \dots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \dots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \dots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{llllllll} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{llllllll} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Compare spaced-word *frequency vectors of sequences*

Spaced Words - recap

1. approach: consider *spaced-word frequencies* for pre-defined pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Spaced-word frequencies)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lllllllll} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Compare spaced-word *frequency vectors of sequences*

Rough measure of sequence dissimilarity; spaced words statistically more stable than contiguous words.

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \dots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lllllllll} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \dots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

S_1	A	T	T	A	C	C	A	C
S_2	A	C	T	A	C	C	G	

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \dots, P_m\}$

Example (Number N of spaced-word matches)

For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Use N to estimate number of substitutions between sequences.

Spaced Words - recap

2. approach: consider *number of spaced-word matches* for pattern set $\mathcal{P} = \{P_1, \ldots, P_m\}$

Example (Number N of spaced-word matches)

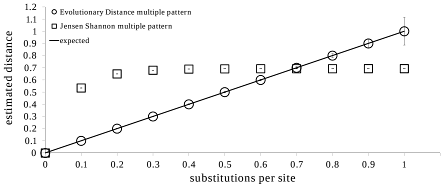
For $\mathcal{P} = \{1101; 1011\}$ and sequences

$$\begin{array}{lcccccccc} S_1 & A & T & T & A & C & C & A & C \\ S_2 & A & C & T & A & C & C & G & \end{array}$$

Use N to estimate number of substitutions between sequences.

But: only possible for sequences with small insertions and deletions.

Spaced Words - recap



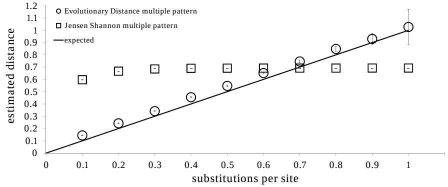
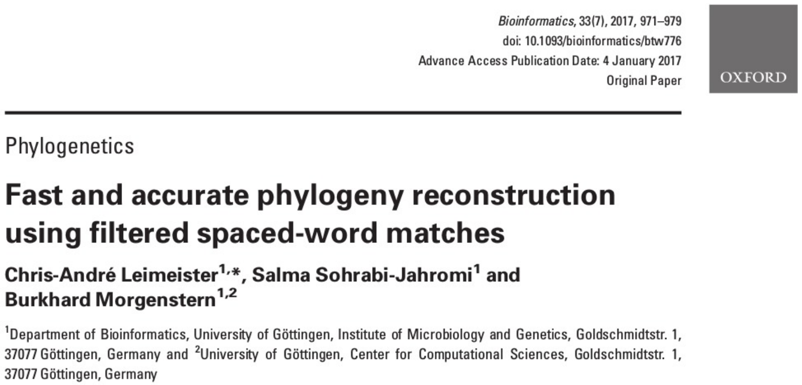


Figure : Estimated distances with *Jensen-Shannon* and new distance measure on simulated DNA sequences (A) without indels (top) and with 1% probability per site (bottom) for multiple spaced words.

Filtered Spaced-Word Matches



Bioinformatics, 33(7), 2017, 971–979
doi: 10.1093/bioinformatics/btw776
Advance Access Publication Date: 4 January 2017
Original Paper

OXFORD

Phylogenetics

Fast and accurate phylogeny reconstruction using filtered spaced-word matches

Chris-André Leimeister[1,*], Salma Sohrabi-Jahromi[1] and Burkhard Morgenstern[1,2]

[1]Department of Bioinformatics, University of Göttingen, Institute of Microbiology and Genetics, Goldschmidtstr. 1, 37077 Göttingen, Germany and [2]University of Göttingen, Center for Computational Sciences, Goldschmidtstr. 1, 37077 Göttingen, Germany

Filtered Spaced-Word Matches

Related approaches:

Filtered Spaced-Word Matches

Related approaches:

(1) *Co-phylog* (Yi *et al.*, 2013)

Filtered Spaced-Word Matches

Related approaches:

(1) *Co-phylog* (Yi *et al.*, 2013)

(2) *andi* (Haubold *et al.*, 2014)

Filtered Spaced-Word Matches

Related approaches:

(1) *Co-phylog* (Yi *et al.*, 2013)

(2) *andi* (Haubold *et al.*, 2014)

Idea: estimate mismatch frequency from local gap-free alignments.

Filtered Spaced-Word Matches

(1) *Co-phylog:*

Filtered Spaced-Word Matches

(1) *Co-phylog:*
Search for pairs of exact word matches of length ℓ, distance one

Filtered Spaced-Word Matches

(1) *Co-phylog:*
Search for pairs of exact word matches of length ℓ, distance one

Example (Co-phylog, $\ell = 4$)

S_1	T	C	A	G	G	A	C	A	T	A	T	C	C	A	T
S_2	A	G	A	C	A	G	A	T	C	C	A	G	C		

Filtered Spaced-Word Matches

(1) *Co-phylog:*
Search for pairs of exact word matches of length ℓ, distance one

Example (Co-phylog, $\ell = 4$)

$$\begin{array}{llllllllllllllll}
S_1 & T & C & A & G & G & A & C & A & T & A & T & C & C & A & T \\
S_2 & A & G & A & C & A & G & A & T & C & C & A & G & C & &
\end{array}$$

Filtered Spaced-Word Matches

(1) *Co-phylog:*
Search for pairs of exact word matches of length ℓ, distance one

Example (Co-phylog, $\ell = 4$)

S_1	T	C	A	G	G	A	C	A	T	A	T	C	C	A	T
S_2	A	G	A	C	A	G	A	T	C	C	A	G	C		

Filtered Spaced-Word Matches

(1) *Co-phylog:*
Search for pairs of exact word matches of length ℓ, distance one

Example (Co-phylog, $\ell = 4$)

S_1	...	G	A	C	A	T	A	T	C	C	...
S_2	...	G	A	C	A	G	A	T	C	C	...

Filtered Spaced-Word Matches

(1) *Co-phylog:*
Search for pairs of exact word matches of length ℓ, distance one

Example (Co-phylog, $\ell = 4$)

S_1	$\ldots$ *G A C A T A T C C* $\ldots$
S_2	$\ldots$ *G A C A G A T C C* $\ldots$

Consider nucleotides between word matches to estimate distances

Filtered Spaced-Word Matches

(2) *andi:*

Filtered Spaced-Word Matches

(2) *andi:*
Search for pairs of maximal exact word matches, same distance in both sequences

Filtered Spaced-Word Matches

(2) *andi:*
Search for pairs of maximal exact word matches, same distance in both sequences

Example (*andi*)

S_1	A	T	C	A	G	G	A	C	A	T	A	C	C	C	C	A	T
S_2	C	G	G	A	C	A	G	A	C	T	C	C	A	G	C		

Filtered Spaced-Word Matches

(2) *andi:*
Search for pairs of maximal exact word matches, same distance in both sequences

Example (*andi*)

S_1	*A*	*T*	*C*	*A*	*G*	*G*	*A*	*C*	*A*	*T*	*A*	*C*	*C*	*C*	*C*	*A*	*T*
S_2	*C*	*G*	*G*	*A*	*C*	*A*	*G*	*A*	*C*	*T*	*C*	*C*	*A*	*G*	*C*		

Filtered Spaced-Word Matches

(2) *andi:*
Search for pairs of maximal exact word matches, same distance in both sequences

Example (*andi*)

S_1	A	T	C	A	G	G	A	C	A	T	A	C	C	C	C	A	T
S_2	C	G	G	A	C	A	G	A	C	T	C	C	A	G	C		

Filtered Spaced-Word Matches

(2) *andi:*
Search for pairs of maximal exact word matches, same distance in both sequences

Example (*andi*)

S_1	...	*G*	*G*	*A*	*C*	*A*	*T*	*A*	*C*	*C*	*C*	*C*	*A*	...
S_2	...	*G*	*G*	*A*	*C*	*A*	*G*	*A*	*C*	*T*	*C*	*C*	*A*	...

Filtered Spaced-Word Matches

(2) *andi:*
Search for pairs of maximal exact word matches, same distance in both sequences

Example (*andi*)

S_1	...	*G*	*G*	*A*	*C*	*A*	*T*	*A*	*C*	*C*	*C*	*C*	*A*	...
S_2	...	*G*	*G*	*A*	*C*	*A*	*G*	*A*	*C*	*T*	*C*	*C*	*A*	...

Consider nucleotides between word matches to estimate distances

Filtered Spaced-Word Matches

Difficulty:

Only *homologous* matches can be used to estimate phylogenetic distances

Filtered Spaced-Word Matches

Difficulty:

Only *homologous* matches can be used to estimate phylogenetic distances

$\Rightarrow$ *Co-phylog* and *andi* use word matches of sufficient length to exclude random similarities.

Filtered Spaced-Word Matches

Difficulty:

Only *homologous* matches can be used to estimate phylogenetic distances

$\Rightarrow$ *Co-phylog* and *andi* use word matches of sufficient length to exclude random similarities.

But: $O(n)$ homologue matches, $O(n^2)$ background matches.

$\Rightarrow$ long word matches necessary if long sequences compared

Filtered Spaced-Word Matches

(3) *FSWM:*

Filtered Spaced-Word Matches

(3) *FSWM:*
Search for spaced-word matches *w.r.t.* given pattern *P*

Filtered Spaced-Word Matches

(3) *FSWM:*
Search for spaced-word matches *w.r.t.* given pattern P

Example ($FSWM$, $P = 11010001$)

S_1 A T C A G G A C A T A C G C C A T
S_2 C G G A C A T G C T C C A G C

Filtered Spaced-Word Matches

(3) *FSWM:*
Search for spaced-word matches *w.r.t.* given pattern *P*

Example (*FSWM*, $P = 11010001$)

S_1	*A*	*T*	*C*	*A*	*G*	*G*	*A*	*C*	*A*	*T*	*A*	*C*	*G*	*C*	*C*	*A*	*T*
S_2	*C*	*G*	*G*	*A*	*C*	*A*	*T*	*G*	*C*	*T*	*C*	*C*	*A*	*G*	*C*		

Filtered Spaced-Word Matches

(3) *FSWM:*
Search for spaced-word matches *w.r.t.* given pattern P

Example (*FSWM*, $P = 11010001$)

S_1	A	T	C	A	G	G	A	C	A	T	A	C	G	C	C	A	T
S_2	C	G	G	A	C	A	T	G	C	T	C	C	A	G	C		

Filtered Spaced-Word Matches

(3) *FSWM:*
Search for spaced-word matches *w.r.t.* given pattern P

Example (*FSWM*, $P = 11010001$)

S_1	...	A	C	A	T	A	C	G	C	...
S_2	...	A	C	A	T	G	C	T	C	...
		1	1	0	1	0	0	0	1	

Filtered Spaced-Word Matches

(3) *FSWM:*
Search for spaced-word matches *w.r.t.* given pattern P

Example ($FSWM$, $P = 11010001$)

S_1	...	A	C	A	T	A	C	G	C	...
S_2	...	A	C	A	T	G	C	T	C	...
		1	1	0	1	0	0	0	1	

Consider nucleotides at *don't-care* positions to estimate distances

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

$C \quad A \quad * \quad A \qquad (S_1)$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)
A	G	$*$	C	(S_1)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)
A	G	$*$	C	(S_1)
C	A	$*$	A	(S_2)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)
A	G	$*$	C	(S_1)
C	A	$*$	A	(S_2)
A	G	$*$	C	(S_2)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)
A	G	$*$	C	(S_1)
C	A	$*$	A	(S_2)
A	G	$*$	C	(S_2)
G	A	$*$	A	(S_2)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)
A	G	$*$	C	(S_1)
C	A	$*$	A	(S_2)
A	G	$*$	C	(S_2)
G	A	$*$	A	(S_2)
A	C	$*$	G	(S_2)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)
A	G	$*$	C	(S_1)
C	A	$*$	A	(S_2)
A	G	$*$	C	(S_2)
G	A	$*$	A	(S_2)
A	C	$*$	G	(S_2)
C	A	$*$	A	(S_2)

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

C	A	$*$	A	(S_1)
A	C	$*$	G	(S_1)
C	A	$*$	A	(S_1)
A	G	$*$	C	(S_1)
C	A	$*$	A	(S_2)
A	G	$*$	C	(S_2)
G	A	$*$	A	(S_2)
A	C	$*$	G	(S_2)
C	A	$*$	A	(S_2)

List $\mathcal{L}$ of all spaced words in S_1 and S_2

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Sort $\mathcal{L}$ in lexicographic order

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Finding spaced-word matches

Example (Find spaced-word matches by *sorting*, $P = 1101$)

S_1	C	A	C	A	G	A	C	
S_2	C	A	G	A	C	A	G	A

A	C	$*$	G	(S_1)
A	C	$*$	G	(S_2)
A	G	$*$	C	(S_1)
A	G	$*$	C	(S_2)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_1)
C	A	$*$	A	(S_2)
C	A	$*$	A	(S_2)
G	A	$*$	A	(S_2)

Identical spaced-words in *buckets* of $\mathcal{L}$

Filtered Spaced-Word Matches

Default parameters in *FSWM*:

- Weight $w = 12$
- 100 *don't-care* positions

$\approx 3.43 \cdot 10^5$ homologous spaced-word matches
$\approx 1.56 \cdot 10^6$ background spaced-word matches

Filtered Spaced-Word Matches

Default parameters in *FSWM*:

- Weight $w = 12$
- 100 *don't-care* positions

$\Rightarrow$ Sensitive, but many random background matches

$\approx 3.43 \cdot 10^5$ homologous spaced-word matches
$\approx 1.56 \cdot 10^6$ background spaced-word matches

Filtered Spaced-Word Matches

Default parameters in *FSWM*:

- Weight $w = 12$
- 100 *don't-care* positions

$\Rightarrow$ Sensitive, but many random background matches

Example (Homologous and background SW matches)

Indel-free sequences of length 5 *Mb*, match probability 0.8:

Filtered Spaced-Word Matches

Default parameters in *FSWM*:

- Weight $w = 12$
- 100 *don't-care* positions

$\Rightarrow$ Sensitive, but many random background matches

Example (Homologous and background SW matches)

Indel-free sequences of length 5 *Mb*, match probability 0.8:

$\approx 3.43 \cdot 10^5$ homologous spaced-word matches
$\approx 1.56 \cdot 10^6$ background spaced-word matches

Remove *low-scoring* spaced-word matches

To filter out random background spaced-word matches:

Remove *low-scoring* spaced-word matches

To filter out random background spaced-word matches:

- Use nucleotide substitution matrix (Chiaromonte *et al.*, 2002)

Remove *low-scoring* spaced-word matches

To filter out random background spaced-word matches:

- Use nucleotide substitution matrix
  (Chiaromonte *et al.*, 2002)
- Calculate *score* for each spaced-word match:
  Sum of substitution scores at *don't-care* positions

Remove *low-scoring* spaced-word matches

To filter out random background spaced-word matches:

- Use nucleotide substitution matrix
  (Chiaromonte *et al.*, 2002)
- Calculate *score* for each spaced-word match:
  Sum of substitution scores at *don't-care* positions
- Discard spaced-word matches with score below threshold

Remove *low-scoring* spaced-word matches

$$\begin{array}{ccccc}
 & A & C & G & T \\
A & 91 & -114 & -31 & -123 \\
C & & 100 & -125 & -31 \\
G & & & 100 & -114 \\
T & & & & 91
\end{array}$$

Remove *low-scoring* spaced-word matches

$$\begin{array}{ccccc} & A & C & G & T \\ A & 91 & -114 & -31 & -123 \\ C & & 100 & -125 & -31 \\ G & & & 100 & -114 \\ T & & & & 91 \end{array}$$

Example (Score of spaced-word match, $P = 1100101$)

$$\begin{array}{lcccccccccccc} S_1: & G & C & T & G & T & A & T & A & C & G & T & C \\ S_2: & G & T & A & C & A & C & T & T & A & T & & \end{array}$$

Remove *low-scoring* spaced-word matches

$$\begin{array}{ccccc} & A & C & G & T \\ A & 91 & -114 & -31 & -123 \\ C & & 100 & -125 & -31 \\ G & & & 100 & -114 \\ T & & & & 91 \end{array}$$

Example (Score of spaced-word match, $P = 1100101$)

$$\begin{array}{lcccccccccccc} S_1: & G & C & T & G & T & A & T & A & C & G & T & C \\ S_2: & G & T & A & C & A & C & T & T & A & T & & \end{array}$$

Remove *low-scoring* spaced-word matches

$$\begin{array}{c|cccc}
 & A & C & G & T \\
A & 91 & -114 & -31 & -123 \\
C & & 100 & -125 & -31 \\
G & & & 100 & -114 \\
T & & & & 91
\end{array}$$

Example (Score of spaced-word match, $P = 1100101$)

$$\begin{array}{lccccccccccccc}
S_1: & G & C & T & G & T & A & T & A & C & G & T & C & \\
S_2: & & & & G & T & A & C & A & C & T & T & A & T \\
P: & & & & & 1 & 1 & 0 & 0 & 1 & 0 & 1 & &
\end{array}$$

Remove *low-scoring* spaced-word matches

$$\begin{array}{c|cccc} & A & C & G & T \\ \hline A & 91 & -114 & -31 & -123 \\ C & & 100 & -125 & -31 \\ G & & & 100 & -114 \\ T & & & & 91 \end{array}$$

Example (Score of spaced-word match, $P = 1100101$)

```
S1 :  G  C  T  G  T  A  T  A  C  G  T  C
S2 :           G  T  A  C  A  C  T  T  A  T
P  :              1  1  0  0  1  0  1
```

Nucleotides at *don't-care* positions

Remove *low-scoring* spaced-word matches

$$\begin{array}{ccccc} & A & C & G & T \\ A & 91 & -114 & -31 & -123 \\ C & & 100 & -125 & -31 \\ G & & & 100 & -114 \\ T & & & & 91 \end{array}$$

Example (Score of spaced-word match, $P = 1100101$)

$$\begin{array}{lcccccccccccc} S_1: & G & C & T & G & T & A & T & A & C & G & T & C & \\ S_2: & & & & G & T & A & C & A & C & T & T & A & T \\ P: & & & & & 1 & 1 & 0 & 0 & 1 & 0 & 1 & & \end{array}$$

Score = -31 + 91 -114 = -54

Remove *low-scoring* spaced-word matches

To remove background noise:

Remove *low-scoring* spaced-word matches

To remove background noise:

- Remove spaced words with score below T.

Remove *low-scoring* spaced-word matches

To remove background noise:

- Remove spaced words with score below T.
- Default value $T = 0$

Remove *low-scoring* spaced-word matches

To remove background noise:

- Remove spaced words with score below T.
- Default value $T = 0$

To visualize distribution of spaced-word matches: plot number of spaced word matches against scores
('Spaced-word histogram')

Spaced-word histograms

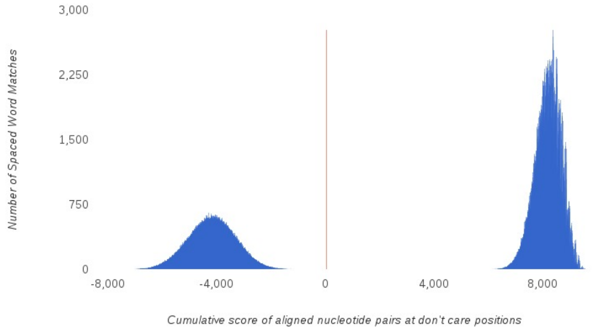


Figure : *i.i.d* sequences, 0.1 subst. per site, indel-free, 5 *Mb*

Spaced-word histograms

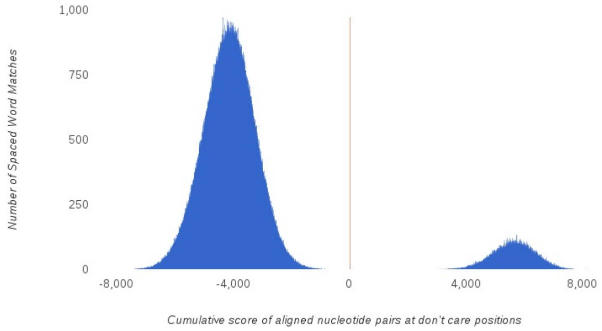


Figure : *i.i.d* sequences, 0.3 subst. per site, indel-free, 5 *Mb*

Spaced-word histograms

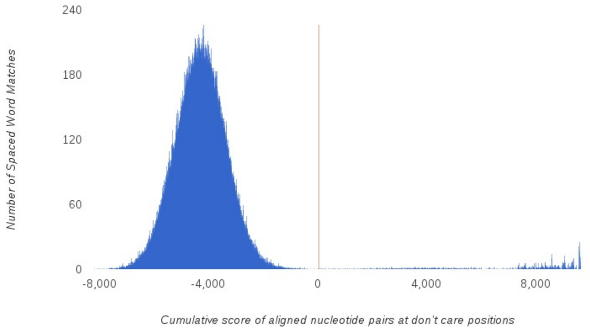


Figure : *Sagittula stellata* E37 vs *Rhodobacterales bacterium* HTCC2255.

Spaced-word histograms

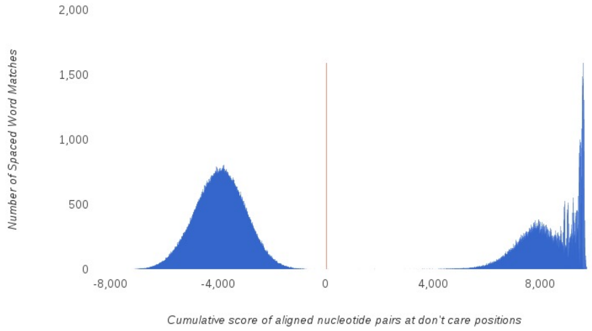


Figure : *Octadecabacter arcticus* 238 vs *Octadecabacter antarticus* 307.

Program Evaluation

Generate pairs of semi-artificial genome sequences:

Program Evaluation

Generate pairs of semi-artificial genome sequences:

- *E. coli K12* as ‘ancestral’ genome

Program Evaluation

Generate pairs of semi-artificial genome sequences:

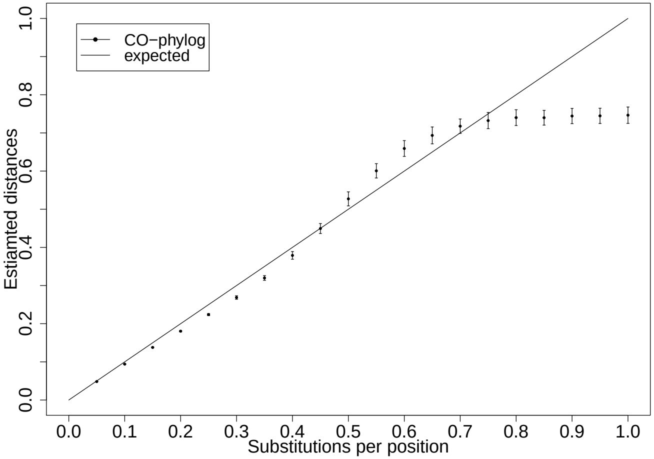
- *E. coli K12* as ‘ancestral’ genome
- Generate substitutions and indels for pairs of ‘descendent’ genomes – between 0 and 1 substitutions per position

Program Evaluation

Generate pairs of semi-artificial genome sequences:

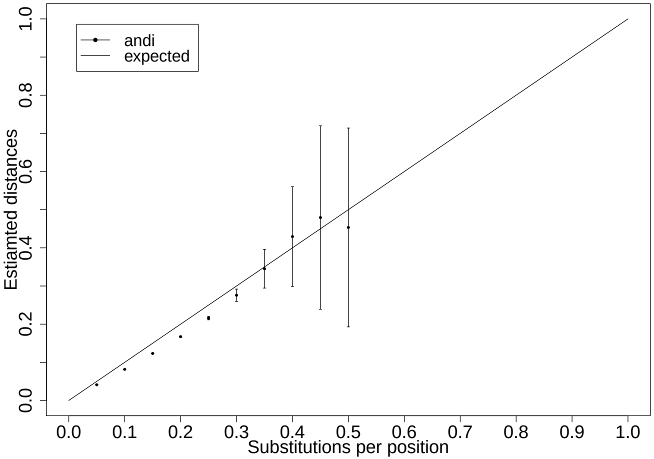
- *E. coli K12* as ‘ancestral’ genome
- Generate substitutions and indels for pairs of ‘descendent’ genomes – between 0 and 1 substitutions per position
- Compare estimated distances to ‘real’ distances

Program Evaluation



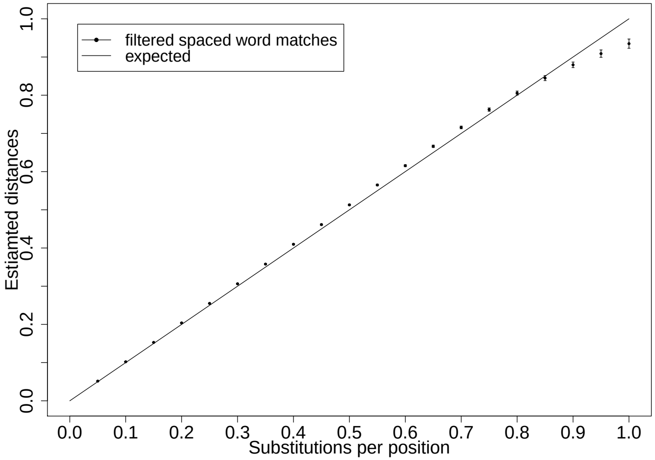
Co-phylog

Program Evaluation



andi

Program Evaluation



FSWM

Program Evaluation

Program Evaluation

- Generate 35 sets of 50 simulated genomes along random tree with *ALF*
  (225-463 *Mb* per data set; ≤ 0.4 substitutions per position)

Program Evaluation

- Generate 35 sets of 50 simulated genomes along random tree with *ALF*
  (225-463 *Mb* per data set; ≤ 0.4 substitutions per position)
- Estimate distances with *Co-phylog*, *andi* and *FSWM*, calculate trees with *Neighbour Joining*

Program Evaluation

- Generate 35 sets of 50 simulated genomes along random tree with *ALF*
  (225-463 *Mb* per data set; ≤ 0.4 substitutions per position)
- Estimate distances with *Co-phylog*, *andi* and *FSWM*, calculate trees with *Neighbour Joining*
- Calculate sum of *Robinson-Foulds* distances

Program Evaluation

- Generate 35 sets of 50 simulated genomes along random tree with *ALF*
  (225-463 *Mb* per data set; ≤ 0.4 substitutions per position)
- Estimate distances with *Co-phylog*, *andi* and *FSWM*, calculate trees with *Neighbour Joining*
- Calculate sum of *Robinson-Foulds* distances

Total sum of *RF* distances:

Co-phylog	446
andi	470
FSWM	424

Program Evaluation

Program Evaluation

Real-world benchmark data: 14 plant genomes (*Brassicales*)

Total size 4.8 Gb, up to 0.63 substitutions per site.

- No reasonable results with *andi*, distance too large
- *Co-phylog* did not finish

Program Evaluation

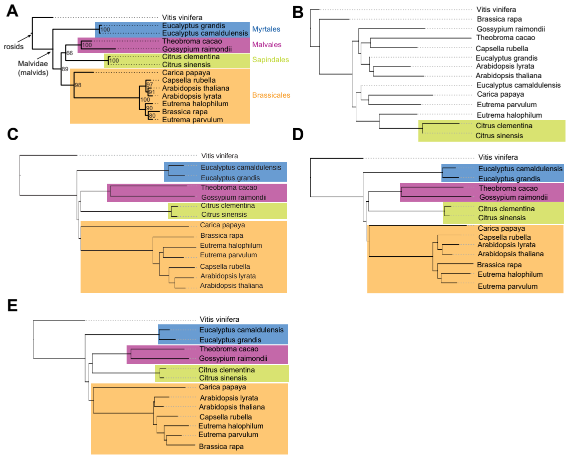


Figure : **A:** Reference tree (protein MSA, Likelihood), **B:** *andi*, **C-E:** FSWM with weight $w = 12, 13, 14$.

Program Evaluation

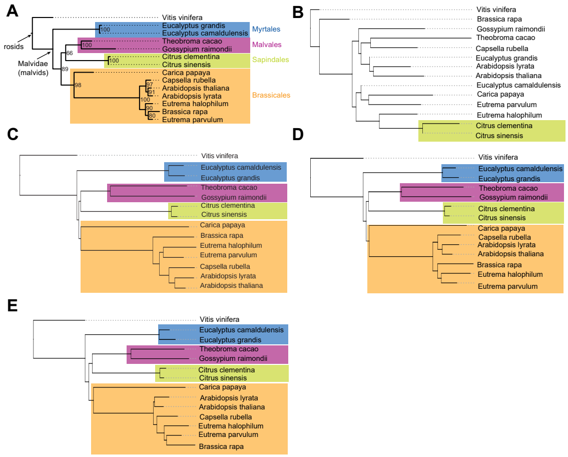
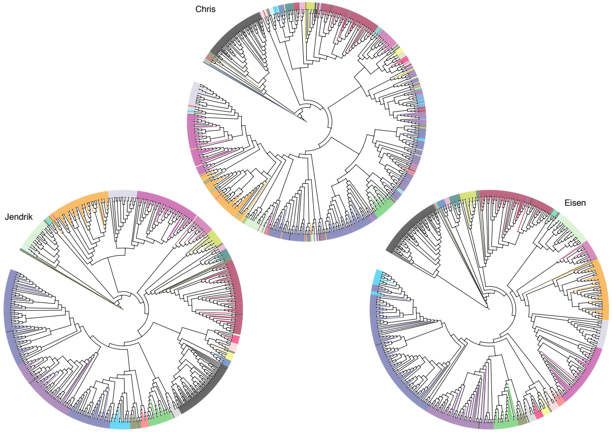


Figure : **A:** Reference tree (protein MSA, Likelihood), **B:** *andi*, **C-E:** FSWM with weight $w = 12, 13, 14$.

Filtered Spaced-Word Matches

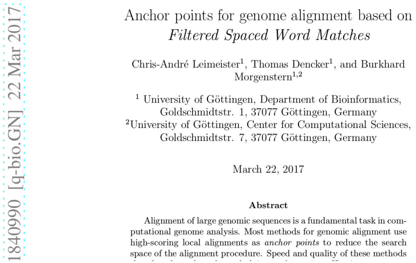
Ongoing project: *Filtered Spaced Word Matches* for protein sequences (Jendrik Schellhorn)



Svenja Schöbel, Jendrik Schellhorn

Spaced Anchors

New project: Use *filtered spaced word matches* as *anchor points* for genome alignment



1840990 [q-bio.GN] 22 Mar 2017

Anchor points for genome alignment based on
Filtered Spaced Word Matches

Chris-André Leimeister[1], Thomas Dencker[1], and Burkhard Morgenstern[1,2]

[1] University of Göttingen, Department of Bioinformatics, Goldschmidtstr. 1, 37077 Göttingen, Germany
[2]University of Göttingen, Center for Computational Sciences, Goldschmidtstr. 7, 37077 Göttingen, Germany

March 22, 2017

Abstract

Alignment of large genomic sequences is a fundamental task in computational genome analysis. Most methods for genomic alignment use high-scoring local alignments as *anchor points* to reduce the search space of the alignment procedure. Speed and quality of these methods

Manuscript uploaded to *arXiv*, submitted to *OUP Bioinformatics*

Spaced Anchors

Example (Anchored pairwise alignment)

S_1	A	G	C	A	C	G	G	T	C	T	C	G	T
S_2	C	A	C	G	A	T	G	A	T	C	G		

Spaced Anchors

Example (Anchored pairwise alignment)

S_1	A	G	C	A	C	G	G	T	C	T	C	G	T
S_2	C	A	C	G	A	T	G	A	T	C	G		

- Find chain of anchor points (e.g. word matches)

Spaced Anchors

Example (Anchored pairwise alignment)

$$\begin{array}{lccccccccccccc} S_1 & A & G & C & A & C & G & G & T & C & T & C & G & T \\ S_2 & C & A & C & G & A & T & G & A & T & C & G & & \end{array}$$

- Find chain of anchor points (e.g. word matches)

Spaced Anchors

Example (Anchored pairwise alignment)

S_1	A	G	C	A	C	G	G	T	C	–	T	C	G	T
S_2	C	–	–	A	C	G	A	T	G	A	T	C	G	–

- Find chain of anchor points (e.g. word matches)
- Align anchor points

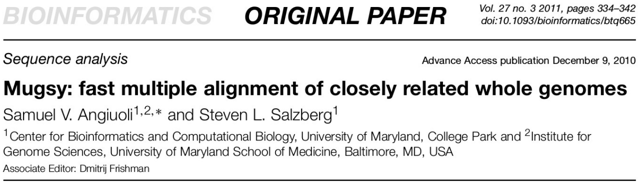
Spaced Anchors

Example (Anchored pairwise alignment)

S_1	A	G	C	A	C	G	–	–	G	T	C	T	C	G	T
S_2	–	–	C	A	C	G	A	T	G	A	–	T	C	G	–

- Find chain of anchor points (e.g. word matches)
- Align anchor points
- Align segments between anchor points

Spaced Anchors



BIOINFORMATICS **ORIGINAL PAPER** *Vol. 27 no. 3 2011, pages 334–342*
doi:10.1093/bioinformatics/btq665

Sequence analysis Advance Access publication December 9, 2010

Mugsy: fast multiple alignment of closely related whole genomes

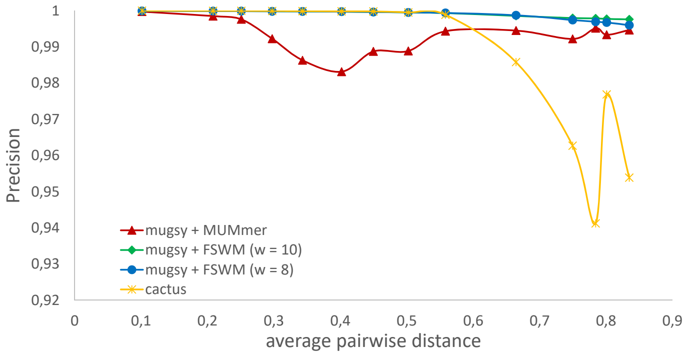
Samuel V. Angiuoli[1,2,*] and Steven L. Salzberg[1]

[1]Center for Bioinformatics and Computational Biology, University of Maryland, College Park and [2]Institute for Genome Sciences, University of Maryland School of Medicine, Baltimore, MD, USA

Associate Editor: Dmitrij Frishman

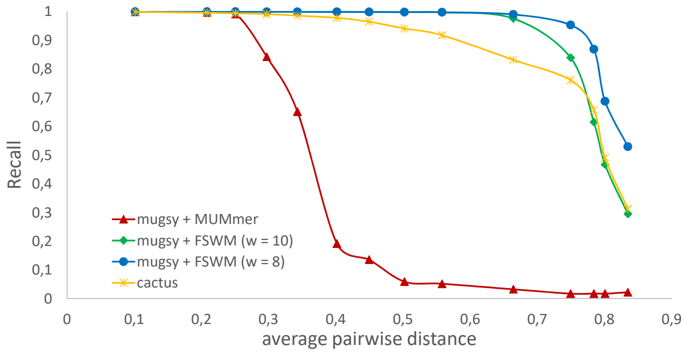
Program evaluation: use *spaced anchors* in *Mugsy* instead of *MUMmer* (exact word matches).

Spaced Anchors



Test results

Spaced Anchors



Test results

Remove *ambiguous* spaced-word matches

Duplicated regions in genomes can confuse phylogeny reconstruction.

Remove *ambiguous* spaced-word matches

Duplicated regions in genomes can confuse phylogeny reconstruction.

Therefore:

FSWM greedily selects one-to-one spaced-word matching

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Spaced word $G * * TA$

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Spaced word $G**TA$

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Spaced word $G**TA$

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Spaced word $G**TA$, 3 times in S_1

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Spaced word $G**TA$, 3 times in S_1

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Spaced word $G * * TA$, 3 times in S_1 , 2 times in S_2

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\Rightarrow$ 6 spaced-word matches involving $G * * TA$

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\Rightarrow$ 6 spaced-word matches involving $G**TA$

	aligned	*score*
$(1,2)$	$GG\ AG$	69

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\Rightarrow$ 6 spaced-word matches involving $G**TA$

	aligned	*score*
$(1,2)$	*GG AG*	69
$(1,9)$	*GG AA*	191

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\Rightarrow$ 6 spaced-word matches involving $G**TA$

	aligned	*score*
$(1,2)$	*GG AG*	69
$(1,9)$	*GG AA*	191
$(6,2)$	*GG GG*	200

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\Rightarrow$ 6 spaced-word matches involving $G**TA$

	aligned	*score*
$(1,2)$	$GG\ AG$	69
$(1,9)$	$GG\ AA$	191
$(6,2)$	$GG\ GG$	200
$(6,9)$	$GG\ GA$	69

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	

$\Rightarrow$ 6 spaced-word matches involving $G**TA$

	aligned	*score*
$(1,2)$	*GG AG*	69
$(1,9)$	*GG AA*	191
$(6,2)$	*GG GG*	200
$(6,9)$	*GG GA*	69
$(8,2)$	*TG AG*	-145

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\Rightarrow$ 6 spaced-word matches involving $G * * TA$

	aligned	*score*
$(1,2)$	*GG AG*	69
$(1,9)$	*GG AA*	191
$(6,2)$	*GG GG*	200
$(6,9)$	*GG GA*	69
$(8,2)$	*TG AG*	-145
$(8,9)$	*TG AA*	-23

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Remove spaced-word matches with negative scores (filtering)

	aligned	*score*
$(1, 2)$	*GG AG*	69
$(1, 9)$	*GG AA*	191
$(6, 2)$	*GG GG*	200
$(6, 9)$	*GG GA*	69
$(8, 2)$	*TG AG*	− 145
$(8, 9)$	*TG AA*	− 23

	A	*C*	*G*	*T*
A	91	−114	−31	−123
C		100	−125	−31
G			100	−114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Remove spaced-word matches with negative scores (filtering)

	aligned	*score*
$(1,2)$	$GG\ AG$	69
$(1,9)$	$GG\ AA$	191
$(6,2)$	$GG\ GG$	200
$(6,9)$	$GG\ GA$	69

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

For one-to-one mapping: sort spaced-word matches . . .

	aligned	*score*
$(1,2)$	*GG AG*	69
$(1,9)$	*GG AA*	191
$(6,2)$	*GG GG*	200
$(6,9)$	*GG GA*	69

	A	*C*	*G*	*T*
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

For one-to-one mapping: sort spaced-word matches . . .

	aligned	*score*
(6, 2)	*GG GG*	200
(1, 9)	*GG AA*	191
(1, 2)	*GG AG*	69
(6, 9)	*GG GA*	69

	A	*C*	*G*	*T*
A	91	−114	−31	−123
C		100	−125	−31
G			100	−114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

. . . use greedy algorithm

	aligned	*score*
$(6,2)$	*GG GG*	200
$(1,9)$	*GG AA*	191
$(1,2)$	*GG AG*	69
$(6,9)$	*GG GA*	69

	A	C	G	T
A	91	-114	-31	-123
C		100	-125	-31
G			100	-114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	*G*	*G*	*A*	*T*	*A*	*G*	*G*	*G*	*T*	*A*	*T*	*A*	*T*	*T*	*A*
S_2 :	*A*	*G*	*G*	*G*	*T*	*A*	*A*	*C*	*G*	*G*	*A*	*T*	*A*	*T*	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

. . . use greedy algorithm

	aligned	*score*	
$(6, 2)$	*GG GG*	200	✓
$(1, 9)$	*GG AA*	191	
$(1, 2)$	*GG AG*	69	
$(6, 9)$	*GG GA*	69	

	A	*C*	*G*	*T*
A	91	−114	−31	−123
C		100	−125	−31
G			100	−114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

... use greedy algorithm

	aligned	*score*	
(6, 2)	*GG GG*	200	✓
(1, 9)	*GG AA*	191	✓
(1, 2)	*GG AG*	69	
(6, 9)	*GG GA*	69	

	A	*C*	*G*	*T*
A	91	−114	−31	−123
C		100	−125	−31
G			100	−114
T				91

Remove *ambiguous* spaced-word matches

Example ($P = 10011$)

S_1 :	G	G	A	T	A	G	G	G	T	A	T	A	T	T	A
S_2 :	A	G	G	G	T	A	A	C	G	G	A	T	A	T	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Result: two spaced-word matches involving $G{*}{*}TA$ accepted

	aligned	*score*	
$(6, 2)$	*GG GG*	200	✓
$(1, 9)$	*GG AA*	191	✓
$(1, 2)$	*GG AG*	69	
$(6, 9)$	*GG GA*	69	

	A	*C*	*G*	*T*
A	91	−114	−31	−123
C		100	−125	−31
G			100	−114
T				91

Introduction

Introduction

Two different approaches to alignment-free sequence comparison:

Introduction

Two different approaches to alignment-free sequence comparison:

- Use words of length k or ‘spaced words’ with fixed underlying patterns P

Introduction

Two different approaches to alignment-free sequence comparison:

- Use words of length k or ‘spaced words’ with fixed underlying patterns P
- Calculate average length of common substrings

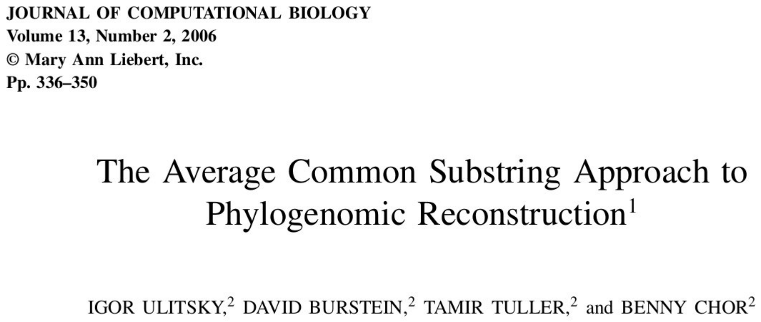
Introduction

Two different approaches to alignment-free sequence comparison:

- Use words of length k or ‘spaced words’ with fixed underlying patterns P
- Calculate average length of common substrings

Advantage of longest-substring methods: get rid of parameter k!

Average Common Substring (ACS)



JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 13, Number 2, 2006
© Mary Ann Liebert, Inc.
Pp. 336–350

The Average Common Substring Approach to Phylogenomic Reconstruction[1]

IGOR ULITSKY,[2] DAVID BURSTEIN,[2] TAMIR TULLER,[2] and BENNY CHOR[2]

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

To compare sequences S_1 and S_2:

For each i in S_1, calculate *longest substring* starting at i matching a substring in S_2.

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	T	A	A	

Average Common Substring (ACS)

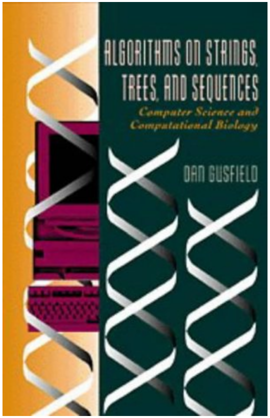
How to find longest substring in S_2 that matches substring starting at position i in S_1?

Average Common Substring (ACS)

How to find longest substring in S_2 that matches substring starting at position i in S_1?

Use *generalized suffix trees!*

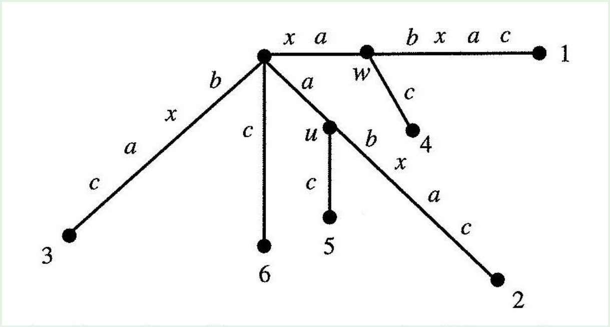
Average Common Substring (ACS)


ALGORITHMS ON STRINGS, TREES, AND SEQUENCES
Computer Science and Computational Biology
DAN GUSFIELD

D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*

Average Common Substring (ACS)

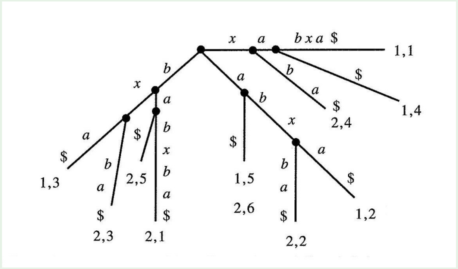
Example (Suffix tree)



Suffix tree for $S = xabxac$ (D. Gusfield)

Average Common Substring (ACS)

Example (Generalized suffix tree)



Generalized suffix tree for strings $S_1 = xabxa$ and $S_2 = babxba$ (D. Gusfield)

Average Common Substring (ACS)

Define distance between sequences S_1 and S_2:

$$
\begin{aligned}
L(S_1, S_2) &:= \text{average length of the longest substring starting} \\
& \text{at } i \text{ in } S_1\text{, matching a subsequence of } S_2 \\
d(S_1, S_2) &:= \frac{\log(|S_2|)}{L(S_1, S_2)} - \frac{\log(|S_1|)}{L(S_1, S_1)} \\
D(S_1, S_2) &:= \frac{d(S_1, S_2) + d(S_2, S_1)}{2}
\end{aligned}
$$

Average Common Substring (ACS)

Define distance between sequences S_1 and S_2:

$$L(S_1, S_2) := \text{average length of the longest substring starting at } i \text{ in } S_1\text{, matching a subsequence of } S_2$$

$$d(S_1, S_2) := \frac{\log(|S_2|)}{L(S_1, S_2)} - \frac{\log(|S_1|)}{L(S_1, S_1)}$$

$$D(S_1, S_2) := \frac{d(S_1, S_2) + d(S_2, S_1)}{2}$$

Note: $D(S_1, S_2)$ not based on stochastic model of evolution!

Average Common Substring (ACS)

Program evaluation:

Average Common Substring (ACS)

Program evaluation:

- No direct evaluation of produced distances!

Average Common Substring (ACS)

Program evaluation:

- No direct evaluation of produced distances!
- Indirect evaluation:

Average Common Substring (ACS)

Program evaluation:

- No direct evaluation of produced distances!
- Indirect evaluation:
  - For set of sequences, calculate pairwise distances

Average Common Substring (ACS)

Program evaluation:

- No direct evaluation of produced distances!
- Indirect evaluation:
    - For set of sequences, calculate pairwise distances
    - Construct tree with *Neighbour-Joining*

Average Common Substring (ACS)

Program evaluation:

- No direct evaluation of produced distances!
- Indirect evaluation:
  - For set of sequences, calculate pairwise distances
  - Construct tree with *Neighbour-Joining*
  - Compare resulting tree to reference trees

Average Common Substring (ACS)

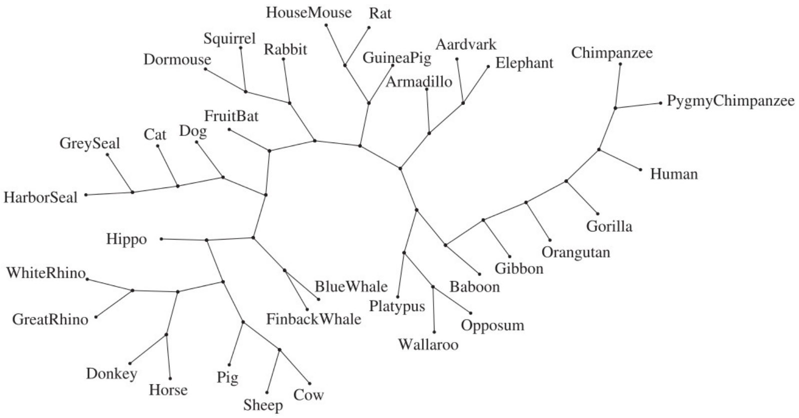


Figure : *ACS* tree based on complete mammalian mtDNA

Average Common Substring (ACS)

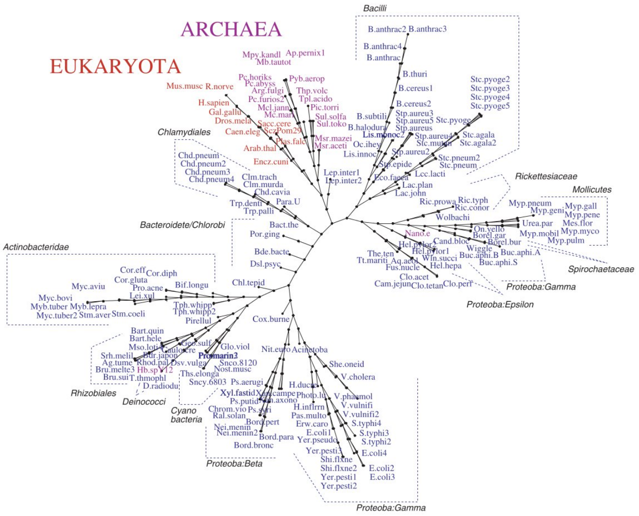
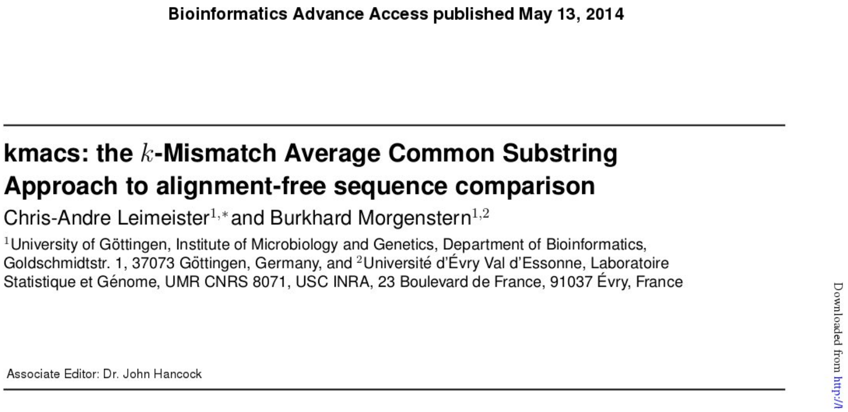


Figure : *ACS* tree based on proteomes

The *kmacs* approach

The *kmacs* approach



Bioinformatics Advance Access published May 13, 2014

kmacs: the k-Mismatch Average Common Substring Approach to alignment-free sequence comparison

Chris-Andre Leimeister[1,*] and Burkhard Morgenstern[1,2]

[1]University of Göttingen, Institute of Microbiology and Genetics, Department of Bioinformatics, Goldschmidtstr. 1, 37073 Göttingen, Germany, and [2]Université d'Évry Val d'Essonne, Laboratoire Statistique et Génome, UMR CNRS 8071, USC INRA, 23 Boulevard de France, 91037 Évry, France

Associate Editor: Dr. John Hancock

The *kmacs* approach

General idea:

The *kmacs* approach

General idea:

Instead of *exact* matches, allow mismatches:

The *kmacs* approach

General idea:

Instead of *exact* matches, allow mismatches:

For each position i in S_1, find longest substring starting at i matching a substring of S_2 with k mismatches.

The *kmacs* approach

Example (Longest k-mismatch common substring)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

Longest string starting at $i = 4$ in S_1 matching a substring of S_2 with $k = 3$ mismatches, length = 11.

The *kmacs* approach

Example (Longest k-mismatch common substring)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

Longest string starting at $i = 4$ in S_1 matching a substring of S_2 with $k = 3$ mismatches, length = 11.

The *kmacs* approach

Example (Longest k-mismatch common substring)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

Longest string starting at $i = 4$ in S_1 matching a substring of S_2 with $k = 3$ mismatches, length = 11.

The *kmacs* approach

Example (Longest k-mismatch common substring)

S_1	T	G	C	A	G	A	C	G	C	A	T
S_2	T	G	G	A	G	T	C	A	C	A	T

Longest string starting at $i = 4$ in S_1 matching a substring of S_2 with $k = 3$ mismatches, length = 11.

The *kmacs* approach

Time complexity for exact solution:

The *kmacs* approach

Time complexity for exact solution:

- Naive algorithm: $O(n^3)$

The *kmacs* approach

Time complexity for exact solution:

- Naive algorithm: $O(n^3)$
- With suffix trees: $O(n^2 \cdot k)$

The *kmacs* approach

Heuristic to approximate longest k-mismatch substring:

The *kmacs* approach

Heuristic to approximate longest k-mismatch substring:

- For each position i in S_i, find longest substring matching substring of S_2 (like in *ACS*)

The *kmacs* approach

Heuristic to approximate longest k-mismatch substring:

- For each position i in S_i, find longest substring matching substring of S_2 (like in *ACS*)
- Extend after first mismatch etc. until $k + 1$th mismatch.

The *kmacs* approach

Example (Heuristic in *kmacs*, $k = 3$)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

The *kmacs* approach

Example (Heuristic in *kmacs*, $k = 3$)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

For position $i = 4$ in S_1

The *kmacs* approach

Example (Heuristic in *kmacs*, $k = 3$)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

For position $i = 4$ in S_1

Find longest matching substring in S_2

The *kmacs* approach

Example (Heuristic in *kmacs*, $k = 3$)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

For position $i = 4$ in S_1

Find longest matching substring in S_2

Extend until $k + 1$-th mismatch

The *kmacs* approach

Example (Heuristic in *kmacs*, $k = 3$)

S_1	T	G	C	A	G	A	C	G	C	A	T
S_2	T	G	G	A	G	T	C	A	C	A	T

For position $i = 4$ in S_1

Find longest matching substring in S_2

Extend until $k + 1$-th mismatch

The *kmacs* approach

Note:

The *kmacs* approach

Note:

- Longest match of substring starting at i may not be unique.

The *kmacs* approach

Note:

- Longest match of substring starting at *i* may not be unique.
- Therefore: extend *all* longest matches to find longest *k*-mismtch substring.

The *kmacs* approach

Example (Longest common substring not unique)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	G	A	T

The *kmacs* approach

Example (Longest common substring not unique)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	G	A	T

For position $i = 2$ in S_1

The *kmacs* approach

Example (Longest common substring not unique)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	G	A	T

For position $i = 2$ in S_1

Several occurrences of longest common substrings in S_2

The *kmacs* approach

Example (Longest common substring not unique)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	G	A	T

For position $i = 2$ in S_1

Several occurrences of longest common substrings in S_2

The *kmacs* approach

Example (Longest common substring not unique)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	G	A	T

For position $i = 2$ in S_1

Several occurrences of longest common substrings in S_2

The *kmacs* approach

Example (Longest common substring not unique)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	G	A	T

For position $i = 2$ in S_1

Several occurrences of longest common substrings in S_2

kmacs extends *all* occurrences, selects longest extension

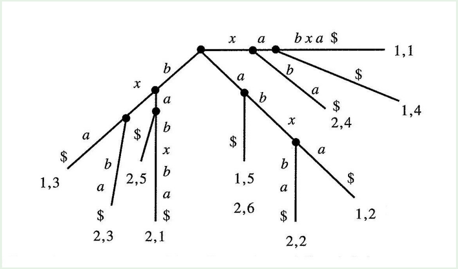
The *kmacs* approach

Generalized suffix trees can be used:

- To find exact word matches (as in *ACS*)
- To extend matches after mismatch

The *kmacs* approach

Example (Generalized suffix tree)



Generalized suffix tree for strings $S_1 = xabxa$ and $S_2 = babxba$ (D. Gusfield, p. 117)

The *kmacs* approach

Time complexity for finding maximal exact matches:

$$O(n \cdot z)$$

The *kmacs* approach

Time complexity for finding maximal exact matches:

$$O(n \cdot z)$$

$z =$ average number of maximal matches to a substring in S_2 starting at a position i in S_1.

The *kmacs* approach

Time complexity for finding maximal exact matches:

$$O(n \cdot z)$$

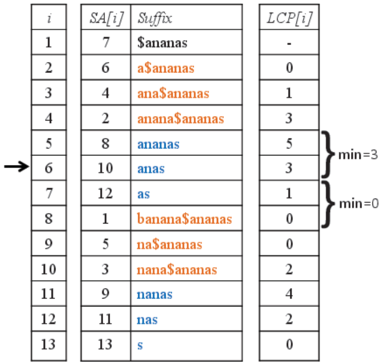
$z =$ average number of maximal matches to a substring in S_2 starting at a position i in S_1.

Time complexity for finding and extending maximal exact matches:

$$O(n \cdot z \cdot k)$$

The *kmacs* approach

Implementation: Use *enhanced suffix arrays* instead of suffix trees (software by Kärkkäinen and Sanders (2003) MPI Saarbrücken)



	i	*SA[i]*	*Suffix*	*LCP[i]*	
	1	7	$ananas	-	
	2	6	a$ananas	0	
	3	4	ana$ananas	1	
	4	2	anana$ananas	3	
	5	8	ananas	5	} min=3
→	6	10	anas	3	
	7	12	as	1	} min=0
	8	1	banana$ananas	0	
	9	5	na$ananas	0	
	10	3	nana$ananas	2	
	11	9	nanas	4	
	12	11	nas	2	
	13	13	s	0	

Figure : Generalized enhanced suffix array for strings `banana` and `ananas`

The *kmacs* approach

As in Ulitsky *et al.* (2006): define distance between S_1 and S_2:

$$
\begin{aligned}
L(S_1, S_2) &:= \text{average length of } k\text{-mismatch longest substrings} \\
d(S_1, S_2) &:= \frac{\log(|S_2|)}{L(S_1, S_2)} - \frac{\log(|S_1|)}{L(S_1, S_1)} \\
D(S_1, S_2) &:= \frac{d(S_1, S_2) + d(S_2, S_1)}{2}
\end{aligned}
$$

Program Evaluation

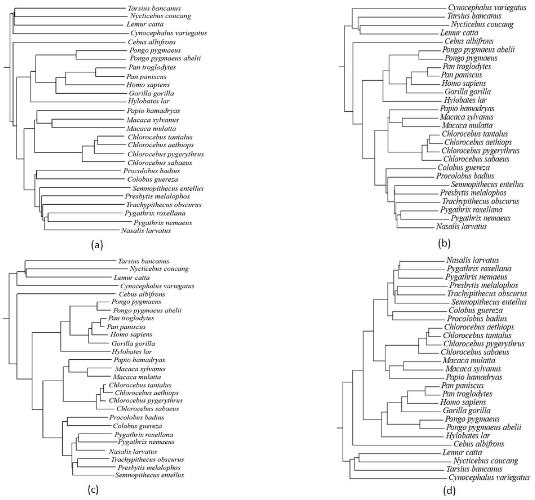


Figure : Mitochondrial DNA sequences (Haubold *et al.*) (a) ACS, (b) tree calculated with *Kr* (Haubold *et al.*), (c) *kmax*, $k = 70$, (d) reference tree.

Program Evaluation

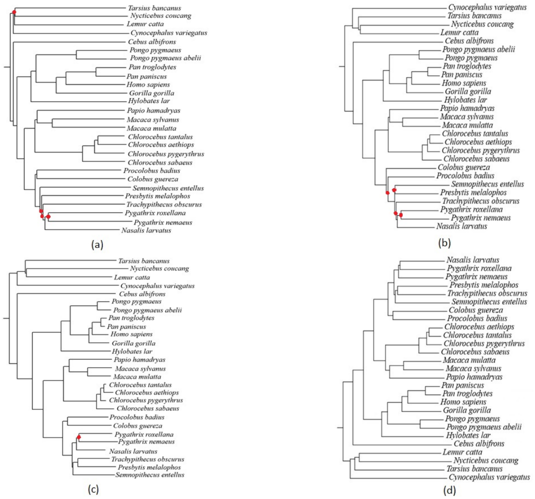


Figure : Mitochondrial DNA sequences (Haubold *et al.*) (a) ACS, (b) tree calculated with *Kr* (Haubold *et al.*), (c) *kmax*, $k = 70$, (d) reference tree.

Program Evaluation

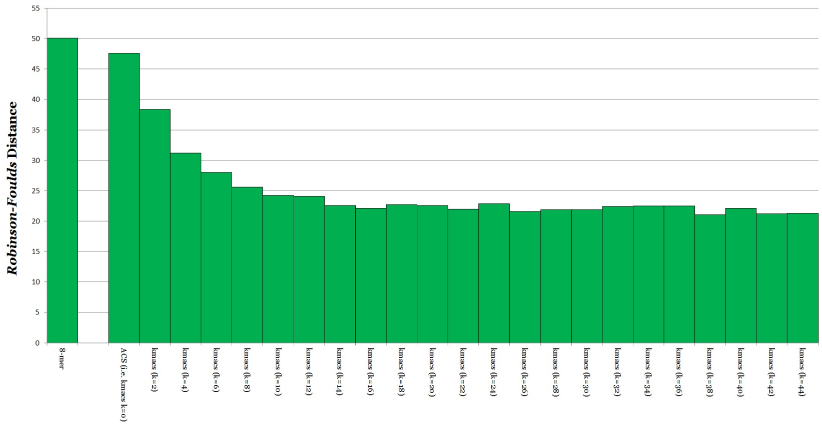


Figure : Simulated DNA sequences (using ROSE). Average *RF* distances for 20 sequence sets with 50 sequences of length 16,000 each. ROSE 'relatednes' = 70

Program Evaluation

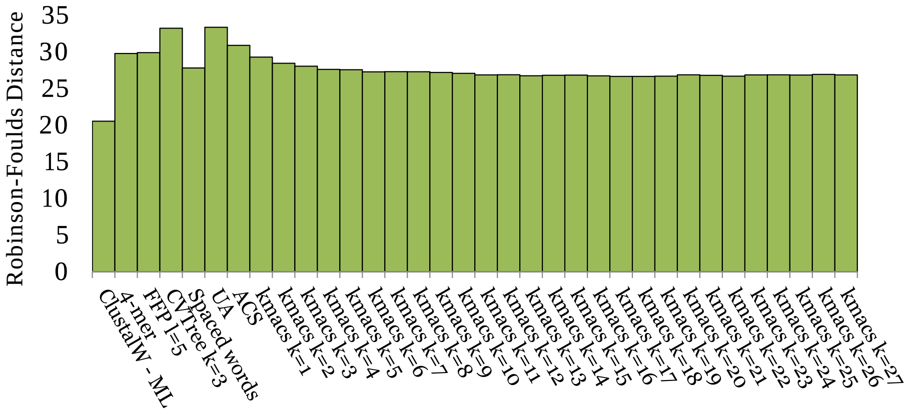


Figure : Results on *BAliBASE* (sum of *RF* distances over 218 Sequence sets)

Program Evaluation

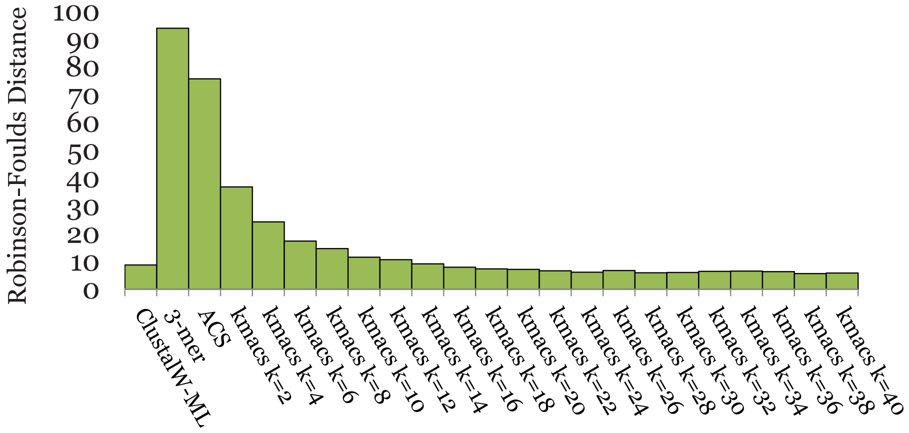


Figure : Simulated protein sequences (using ROSE). Average values for 20 sequence sets with 125 sequences of length 300 each. ROSE ‘relatednes’ = 480

Program Evaluation

Method	runtime (s)
Clustal W	1,817
Clustal Ω	1,039
spaced words, 1 pattern, $k = 8$	0.3
spaced words, 100 patterns, $k = 8$	27.6
ACS	2.8
K_r	0.9
CVTree	0.5
kmacs, $k = 10$	7.6
kmacs, $k = 50$	21.4

Program runtime on 50 simulated DNA sequences of length 16,000.

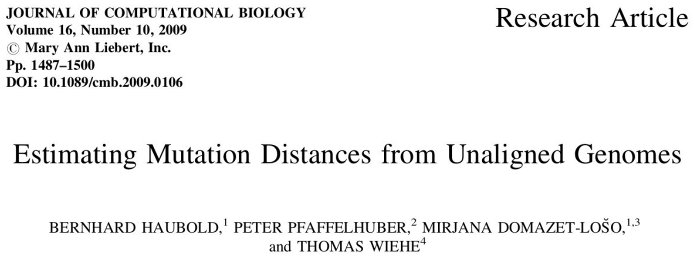
Program Evaluation

Seq. length	k	runtime (s)
100 kb	0	0.04
100 kb	50	0.12
100 kb	100	0.29
1 mb	0	0.19
1 mb	50	1.15
1 mb	100	2.00
10 mb	0	3.11
10 mb	50	13.47
10 mb	100	22.01

Program runtime on pairs of simulated DNA sequences.

K_r (‘shustring’ approach)

First alignment-free approach to estimate number of substitutions per sequence position!



JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 16, Number 10, 2009
© Mary Ann Liebert, Inc.
Pp. 1487–1500
DOI: 10.1089/cmb.2009.0106

Research Article

Estimating Mutation Distances from Unaligned Genomes

BERNHARD HAUBOLD,[1] PETER PFAFFELHUBER,[2] MIRJANA DOMAZET-LOŠO,[1,3] and THOMAS WIEHE[4]

K_r *('shustring' approach)*

To compare S_1 and S_2:

For each i in S_1, calculate *longest unique substring ('shustring')* starting at i (equivalent to finding longest common substring)

K_r (‘shustring’ approach)

To compare S_1 and S_2:

For each i in S_1, calculate *longest unique substring (‘shustring’)* starting at i (equivalent to finding longest common substring)

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	A	T	A	

For position $i = 4$ in S_1

K_r (‘shustring’ approach)

To compare S_1 and S_2:

For each i in S_1, calculate *longest unique substring (‘shustring’)* starting at i (equivalent to finding longest common substring)

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	A	T	A	

For position $i = 4$ in S_1

K_r (*'shustring' approach*)

To compare S_1 and S_2:

For each i in S_1, calculate *longest unique substring ('shustring')* starting at i (equivalent to finding longest common substring)

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	A	T	A	

For position $i = 4$ in S_1

K_r (‘shustring’ approach)

To compare S_1 and S_2:

For each i in S_1, calculate *longest unique substring (‘shustring’)* starting at i (equivalent to finding longest common substring)

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	A	T	A	

For position $i = 4$ in S_1

K_r *(‘shustring’ approach)*

To compare S_1 and S_2:

For each i in S_1, calculate *longest unique substring (‘shustring’)* starting at i (equivalent to finding longest common substring)

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	A	T	A	

For position $i = 4$ in S_1

K_r (*'shustring' approach*)

To compare S_1 and S_2:

For each i in S_1, calculate *longest unique substring ('shustring')* starting at i (equivalent to finding longest common substring)

Example (ACS)

S_1	C	A	T	T	G	G	A	G	T	C	G	T	A
S_2	A	T	G	G	A	G	T	C	A	A	T	A	

For position $i = 4$ in S_1 shustring length = 8

K_r (‘shustring’ approach)

To estimate number d of substitutions per position:

K_r (‘shustring’ approach)

To estimate number d of substitutions per position:

- Calculate *expected* shustring length as function of mismatch rate p

K_r (‘shustring’ approach)

To estimate number d of substitutions per position:

- Calculate *expected* shustring length as function of mismatch rate p
- Moment-based approach: substitute *expected* shustring length by *empirical* average shustring length to calculate p

K_r (*'shustring' approach*)

To estimate number d of substitutions per position:

- Calculate *expected* shustring length as function of mismatch rate p
- Moment-based approach: substitute *expected* shustring length by *empirical* average shustring length to calculate p
- Calculate d from p using *Jukes-Cantor* formula

K_r (*'shustring' approach*)

Definition

Define random variables:

$X_{i,j} =$ *length of longest exact match at* i *and* j*, resp.*

$X_i = \max_{1 \le j \le L} X_{i,j}$

K_r (‘shustring’ approach)

Definition

Define random variables:

$X_{i,j}$ = *length of longest exact match at i and j, resp.*

$X_i = \max_{1 \leq j \leq L} X_{i,j}$

$\rightarrow$ calculate $P(X_i = m)$ and $E(X_i)$

K_r ('shustring' approach)

Results: precise estimation of distances up to ~ 0.5 substitutions per position

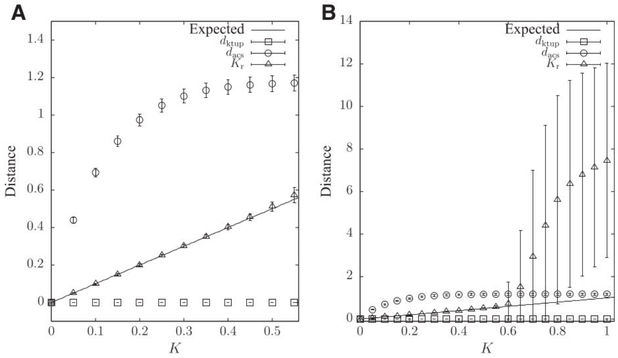


FIG. 2. Pairwise distances as a function of the number of substitutions per site, K. (**A**) Range of substitutions/site (K) values that are well approximated by K_r. (**B**) Range of K values with "phase transition" of K_r. Each symbol represents the mean $\pm$ standard deviation of 10^4 iterations with sequence pairs of length 100 kb each and GC content of 0.5.

The length of *k*-mismatch common substrings

The length of *k*-mismatch common substrings

Morgenstern *et al. Algorithms Mol Biol* (2017) 12:27
https://doi.org/10.1186/s13015-017-0118-8

Algorithms for Molecular Biology

RESEARCH Open Access

Phylogeny reconstruction based on the length distribution of *k*-mismatch common substrings

Burkhard Morgenstern*, Svenja Schöbel and Chris-André Leimeister

Alg. Mol. Biol. 12, 27

The length of k-mismatch common substrings

Generalize idea from Haubold *et al.* (2009) for same model of evolution. Goal: estimate match probability p.

The length of *k*-mismatch common substrings

Generalize idea from Haubold *et al.* (2009) for same model of evolution. Goal: estimate match probability p.

Definition (Length of k-mismatch common substrings)

Define random variables:

$$X_{i,j}^{(k)} = \text{length of longest } k\text{-mismatch common substring at } i, j$$

$$X_{i}^{(k)} = \max_{1 \le j \le L} X_{i,j}^{(k)}$$

The length of k-mismatch common substrings

Remark (Length distribution of $X_{i,j}^{(k)}$)

$$P\left(X_{i,j}^{(k)} = m\right) = \begin{cases} \binom{m}{k} p^{m-k}(1-p)^{k+1} & \text{if } i = j \\ \binom{m}{k} q^{m-k}(1-q)^{k+1} & \text{else} \end{cases} \tag{1}$$

The length of k-mismatch common substrings

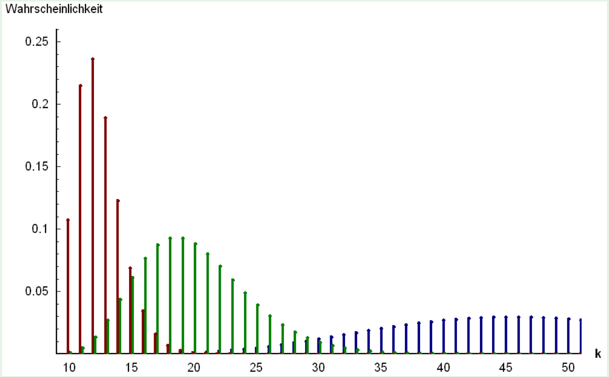
Remark (Length distribution of $X_{i,j}^{(k)}$)

$$P\left(X_{i,j}^{(k)} = m\right) = \begin{cases} \binom{m}{k} p^{m-k}(1-p)^{k+1} & \textit{if } i = j \\ \binom{m}{k} q^{m-k}(1-q)^{k+1} & \textit{else} \end{cases} \tag{1}$$

'Negative binomial' distribution.

The length of *k*-mismatch common substrings

Example (Negative binomial distribution)



Negative binomial distribution for varying values of *p* (Wikipedia)

The length of k-mismatch common substrings

Idea: find longest *exact match* starting at position i in S_1, consider length of *extension* with k mismatches (as in *kmacs* heuristics)

For position $i = 4$ in S_1, $k = 3$

Find longest matching substring in S_2

Extend until $k + 1$-th mismatch

Consider only length of *extension*

The length of k-mismatch common substrings

Idea: find longest *exact match* starting at position i in S_1, consider length of *extension* with k mismatches (as in *kmacs* heuristics)

Example (k-mismatch *extension* of longest exact match)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

The length of k-mismatch common substrings

Idea: find longest *exact match* starting at position i in S_1, consider length of *extension* with k mismatches (as in *kmacs* heuristics)

Example (k-mismatch *extension* of longest exact match)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

For position $i = 4$ in $S_1, k = 3$

The length of k-mismatch common substrings

Idea: find longest *exact match* starting at position i in S_1, consider length of *extension* with k mismatches (as in *kmacs* heuristics)

Example (k-mismatch *extension* of longest exact match)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

For position $i = 4$ in S_1, $k = 3$

Find longest matching substring in S_2

The length of k-mismatch common substrings

Idea: find longest *exact match* starting at position i in S_1, consider length of *extension* with k mismatches (as in *kmacs* heuristics)

Example (k-mismatch *extension* of longest exact match)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

For position $i = 4$ in S_1, $k = 3$

Find longest matching substring in S_2

Extend until $k + 1$-th mismatch

The length of k-mismatch common substrings

Idea: find longest *exact match* starting at position i in S_1, consider length of *extension* with k mismatches (as in *kmacs* heuristics)

Example (k-mismatch *extension* of longest exact match)

S_1	C	A	T	T	G	C	A	G	A	C	G	C	A	T	C
S_2	A	T	G	G	A	G	T	C	A	C	A	T	A	T	T

For position $i = 4$ in S_1, $k = 3$

Find longest matching substring in S_2

Extend until $k + 1$-th mismatch

Consider only length of *extension*

The length of k-mismatch common substrings

Idea: find longest *exact match* starting at position i in S_1, consider length of *extension* with k mismatches (as in *kmacs* heuristics)

Example (k-mismatch *extension* of longest exact match)

S_1	A	G	A	C	G	C	A	T
S_2	A	G	T	C	A	C	A	T

For position $i = 4$ in $S_1, k = 3$

Find longest matching substring in S_2

Extend until $k + 1$-th mismatch

Consider only length of *extension*

The length of k-mismatch common substrings

Definition

The length of *k*-mismatch common substrings

Definition

1. $P_h =$ *probablitiy that longest exact match is 'homologue',* i.e. *matches at same position (in indel-free model)*

The length of *k*-mismatch common substrings

Definition

1. $P_h =$ *probablitiy that longest exact match is ‘homologue’,* i.e. *matches at same position (in indel-free model)*
2. $P_b =$ *probability that longest exact match is* not *‘homologue’*

The length of k-mismatch common substrings

Definition

1. P_h = *probablitiy that longest exact match is 'homologue',* i.e. *matches at same position (in indel-free model)*
2. P_b = *probability that longest exact match is* not *'homologue'*
3. $\hat{X}_i^{(k)}$ = *length of k-mismatch extension at position i in* kmacs *heuristics (running with $k+1$)* after *longest exact match*

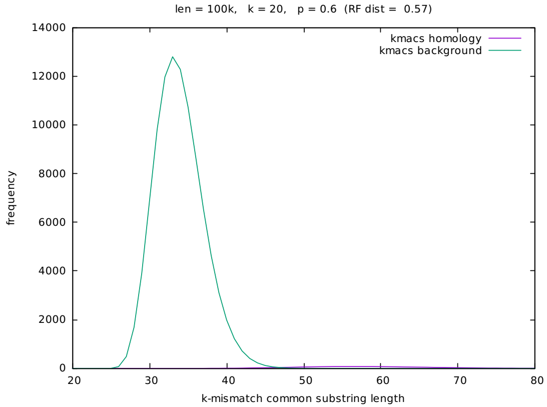
The length of *k*-mismatch common substrings

Then, with (1), we obtain

Theorem (Length distribution of *k*-mismatch extension)

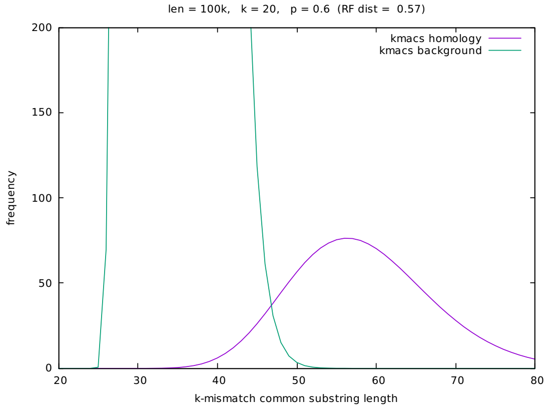
$$P\left(\hat{X}_i^{(k)} = m\right) = P_h \cdot \binom{m}{k} p^{m-k}(1-p)^{k+1} + P_b \cdot \binom{m}{k} q^{m-k}(1-q)^{k+1}$$

The length of *k*-mismatch common substrings



Expected number of k-mismatch common substrings of length m with *kmacs* for sequence length 100 *kb*, $p = 0.6$ and $k = 20$

The length of *k*-mismatch common substrings



Expected number of *k*-mismatch common substrings of length *m* with *kmacs* for sequence length 100 *kb*, $p = 0.6$ and $k = 20$

The length of k-mismatch common substrings

Corollary

1. *The distribution of* $\hat{X}_i^{(k)}$ *is the sum of two* negative binomials *('homolgous' and 'background') with maxima at*

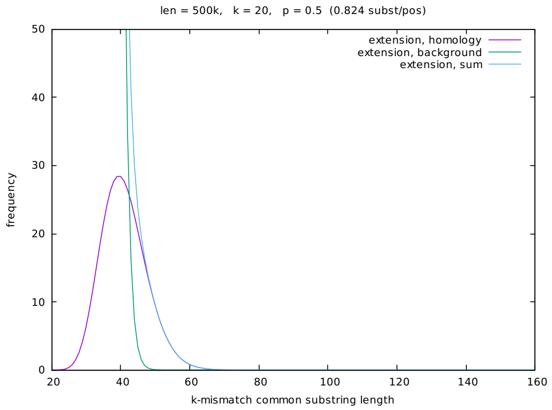
$$\left\lceil \frac{k}{1-p} - 1 \right\rceil \quad \textit{and} \quad \left\lceil \frac{k}{1-q} - 1 \right\rceil$$

2. *If p and k are large enough,* $\hat{X}_i^{(k)}$ *is* bimodal, *and we can estimate*

$$\hat{p} = \frac{m_E + 1 - k}{m_E + 1} \tag{2}$$

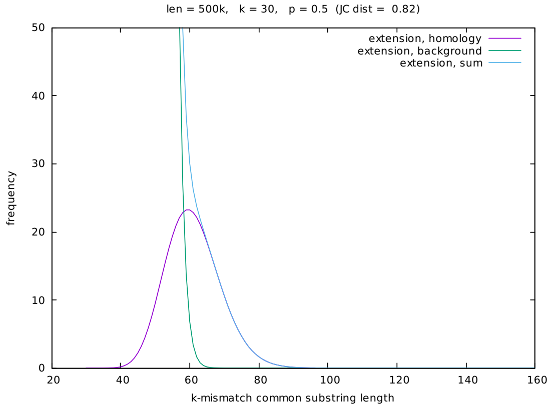
with m_E location of empirical 'homologous' peak.

The length of *k*-mismatch common substrings



Expected number of *k*-mismatch *extensions* for seq. length 500 *kb*
$p = 0.5$ and $k = 20$

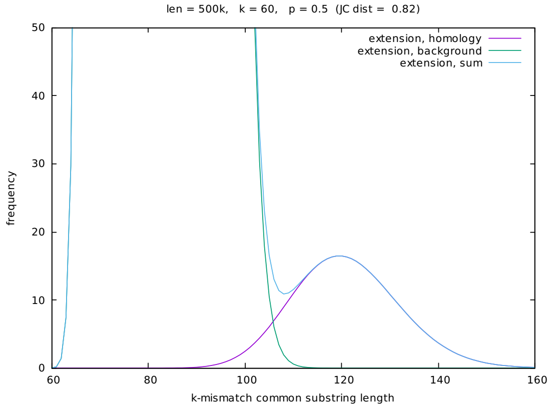
The length of *k*-mismatch common substrings



Expected number of k-mismatch *extensions* for seq. length 500 *kb*

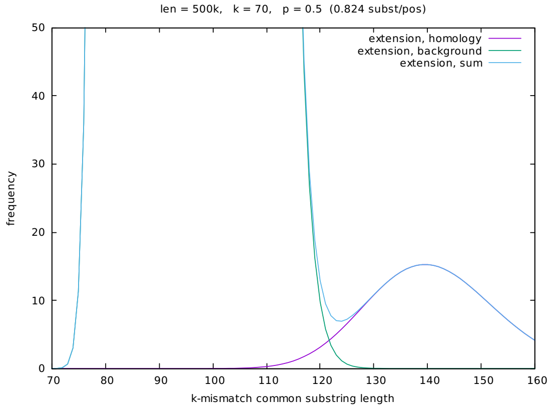
$p = 0.5$ and $k = 30$

The length of *k*-mismatch common substrings



Expected number of *k*-mismatch *extensions* for seq. length 500 *kb*
$p = 0.5$ and $k = 60$

The length of *k*-mismatch common substrings



Expected number of *k*-mismatch *extensions* for seq. length 500 *kb*

$p = 0.5$ and $k = 70$

The length of *k*-mismatch common substrings

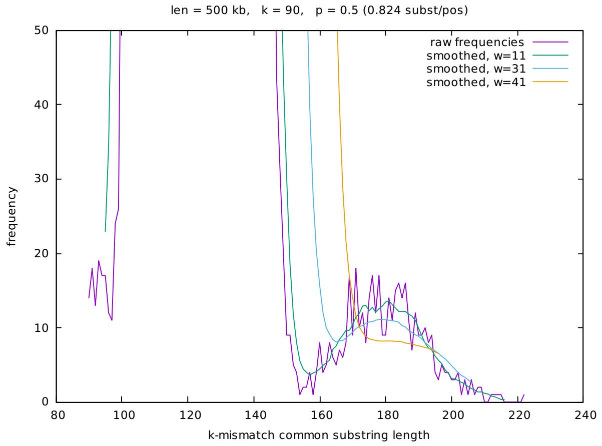
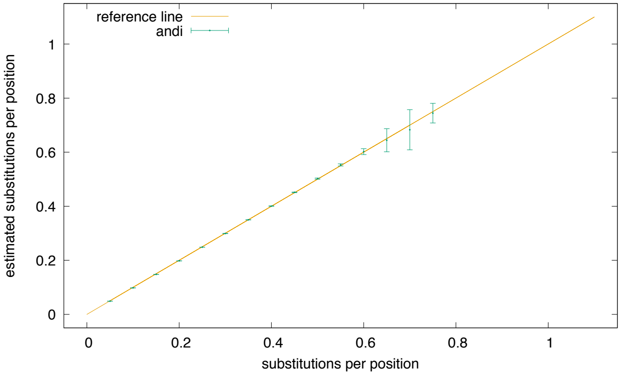


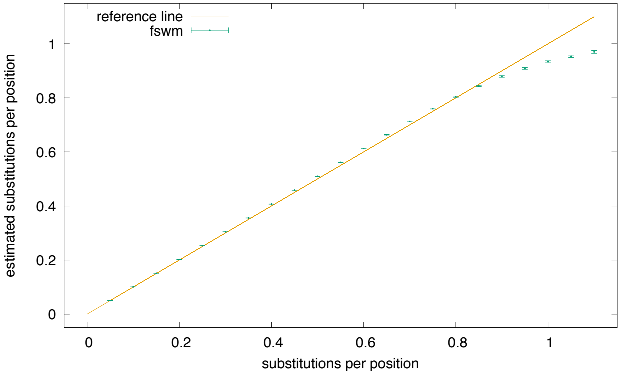
Figure : Empirical number of *k*-mismatch extensions, smoothed with window width 1, 11, 31, 41

The length of *k*-mismatch common substrings



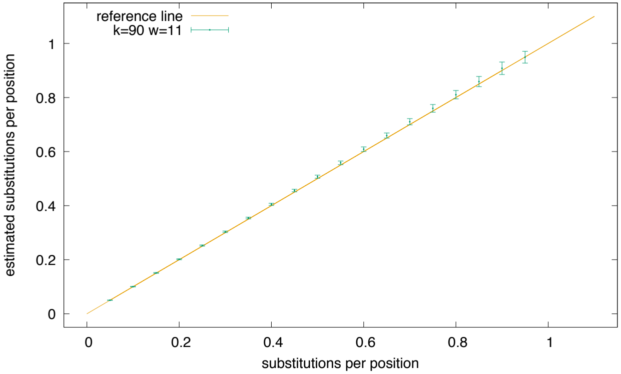
Estimated vs. real distances for simulated sequences, *andi*

The length of *k*-mismatch common substrings



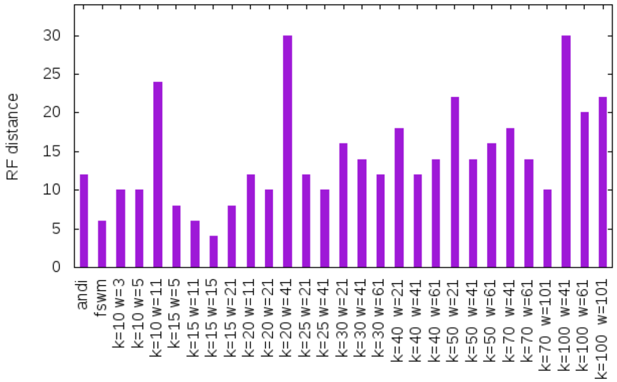
Estimated vs. real distances for simulated sequences, *FSWM*

The length of *k*-mismatch common substrings



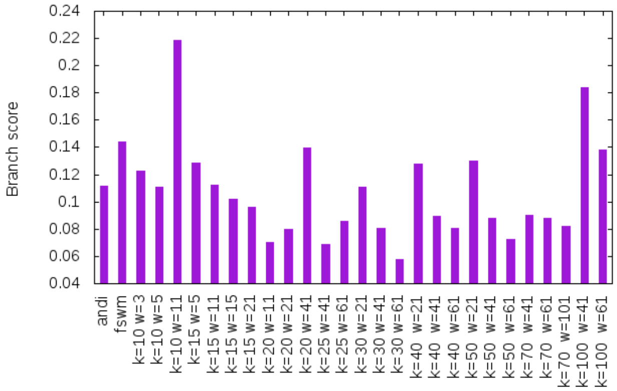
Estimated vs. real distances for simulated sequences based on length of *k*-mismatch common substrings

The length of *k*-mismatch common substrings



Evaluation on 27 mitochondrial genomes from primates
(Robinson-Foulds distance)

The length of *k*-mismatch common substrings



Evaluation on 27 mitochondrial genomes from primates (branch score distance)

The length of k-mismatch common substrings

Ongoing / future projects:

The length of *k*-mismatch common substrings

Ongoing / future projects:

- Better ways of finding second peak in length distribution

The length of *k*-mismatch common substrings

Ongoing / future projects:

- Better ways of finding second peak in length distribution
- Dealing with insertions and deletions

The length of k-mismatch common substrings

Ongoing / future projects:

- Better ways of finding second peak in length distribution
- Dealing with insertions and deletions
- Optimal parameters (k, smoothing window)

The length of *k*-mismatch common substrings

Ongoing / future projects:

- Better ways of finding second peak in length distribution
- Dealing with insertions and deletions
- Optimal parameters (*k*, smoothing window)
- Systematic applications to genome data

Thank you:

Chris Leimeister
Lars Hahn
Marcus Boden
Thomas Dencker
Bingyao Zhu
Jendrik Schellhorn
Svenja Schöbel
Salma Sohrabi-Jahromi
Sanghamitra Bandyopadhyay
Angana Chakraborty
Laurent Noé
Rachid Ounit
Stefano Lonardi