Finding Repetitive Structures in Large Sequences

Jens Stoye

Genome Informatics, Faculty of Technology

and

Institute of Bioinformatics, Center of Biotechnology

Bielefeld University, Germany

Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Finding repetitive structures in large sequences

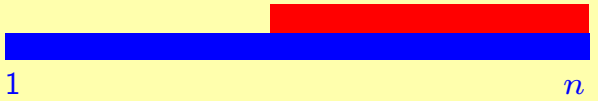
- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Finding repetitive structures in large sequences

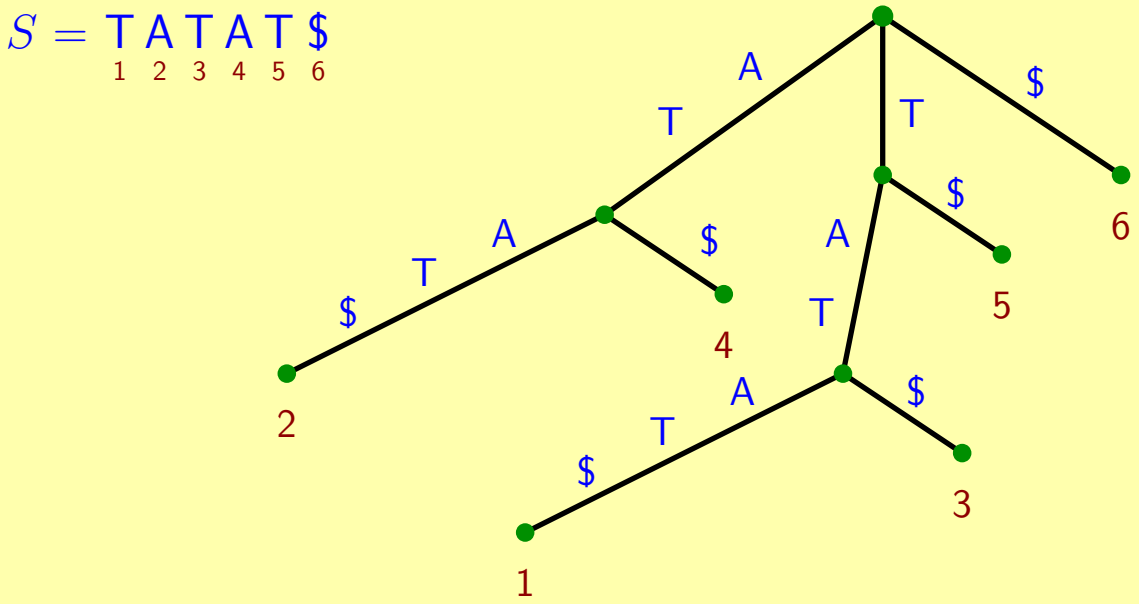
- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Suffix Tree: Definition

- A suffix of a string S of length n
  is a substring of S that ends at position n.

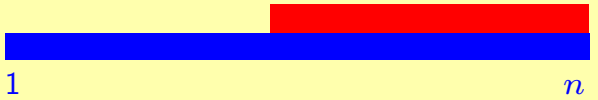


- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.



Suffix Tree: Definition

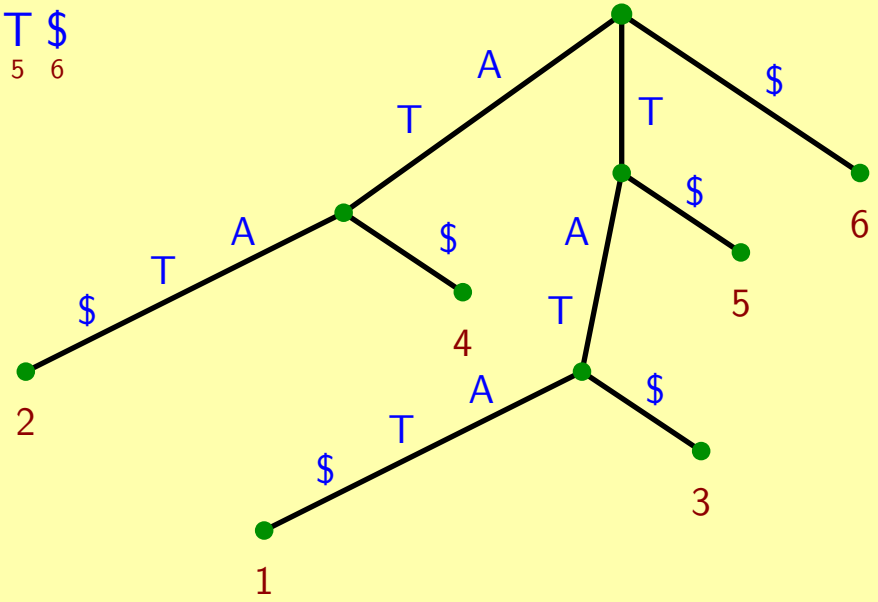
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

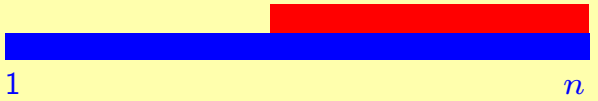
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

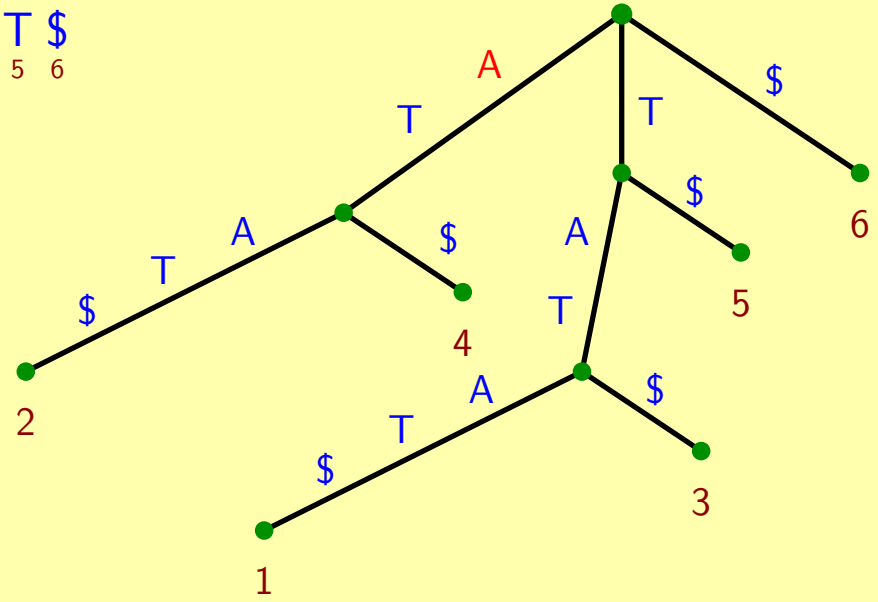
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

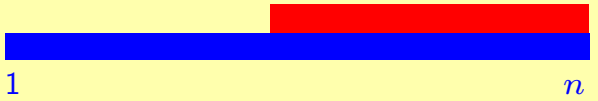
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

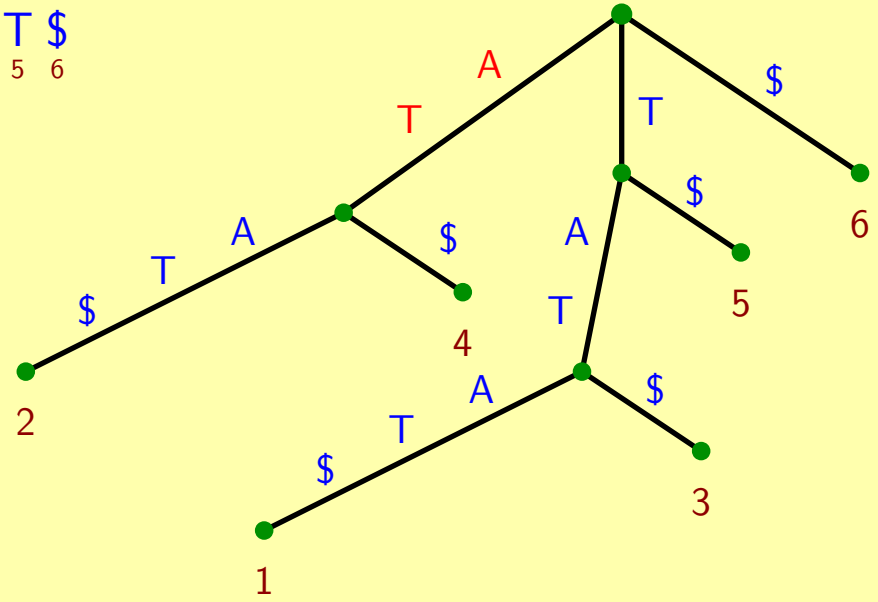
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

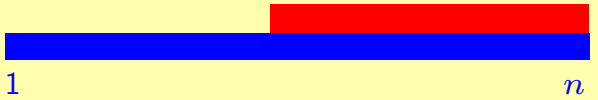
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

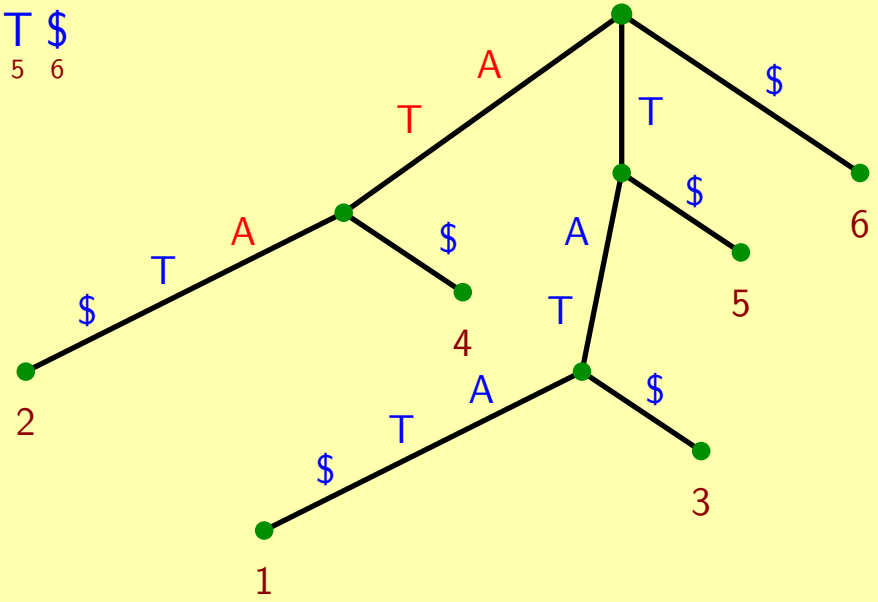
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

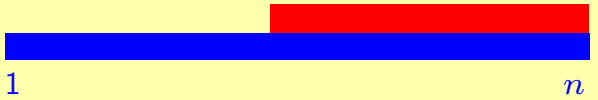
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

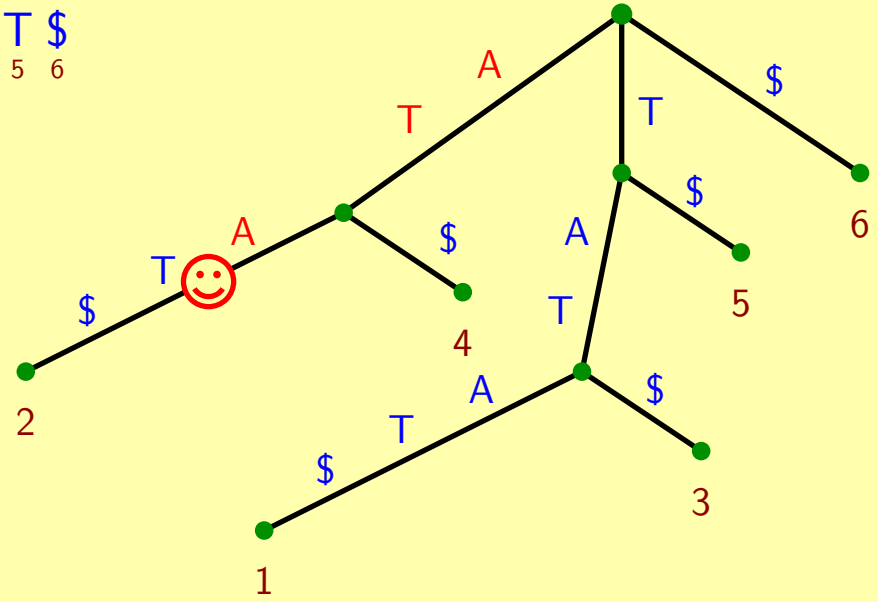
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

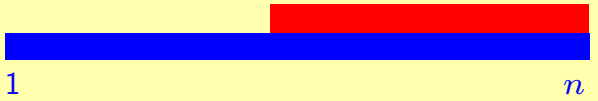
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A

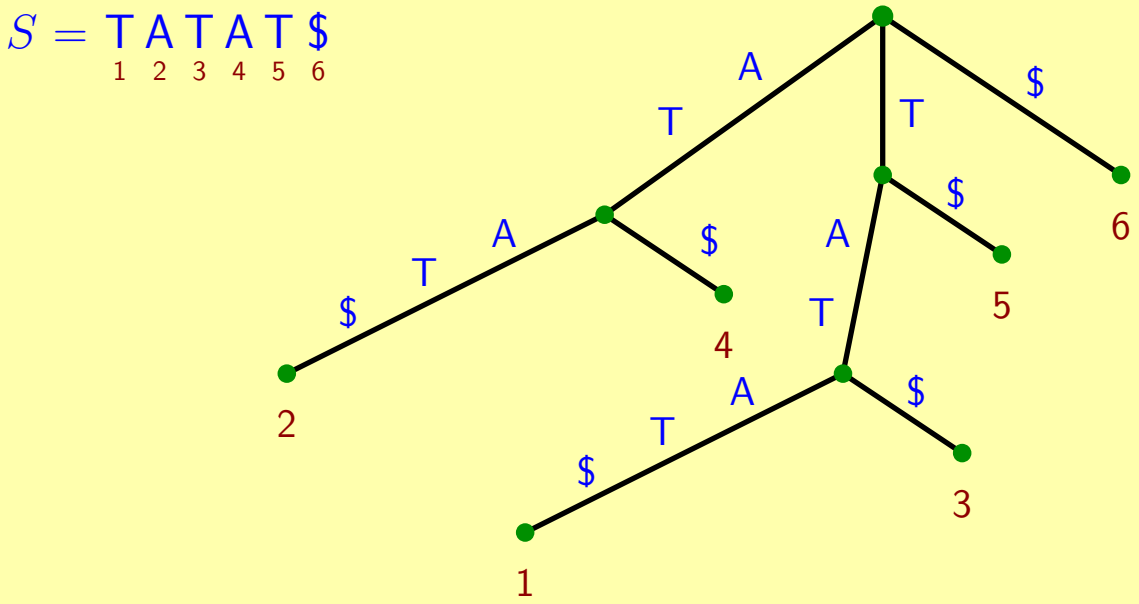


Suffix Tree: Definition

- A suffix of a string S of length n
  is a substring of S that ends at position n.

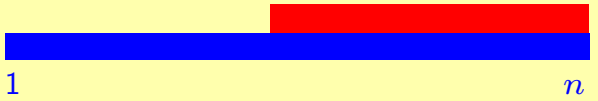


- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.



Suffix Tree: Definition

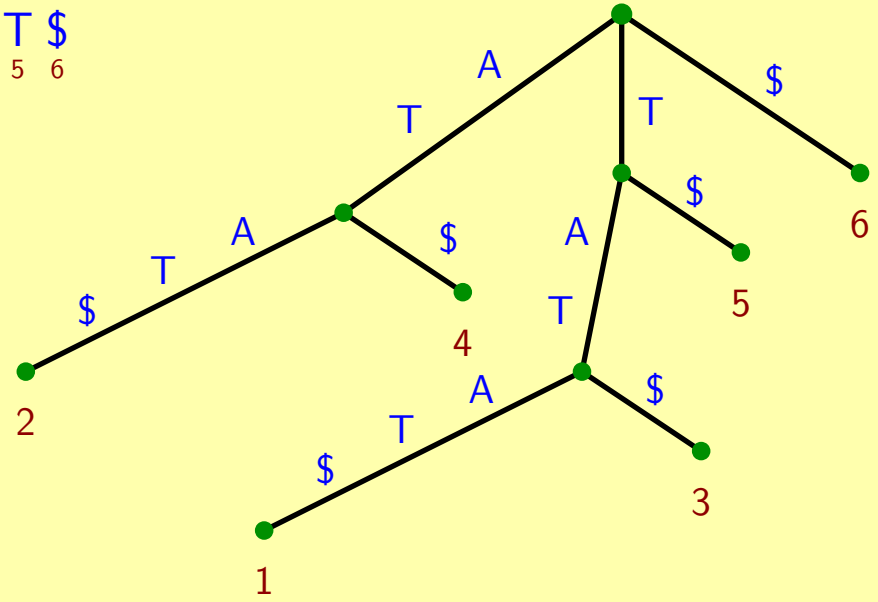
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

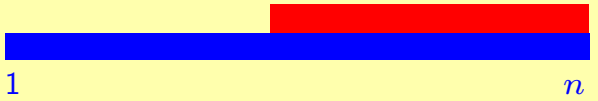
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

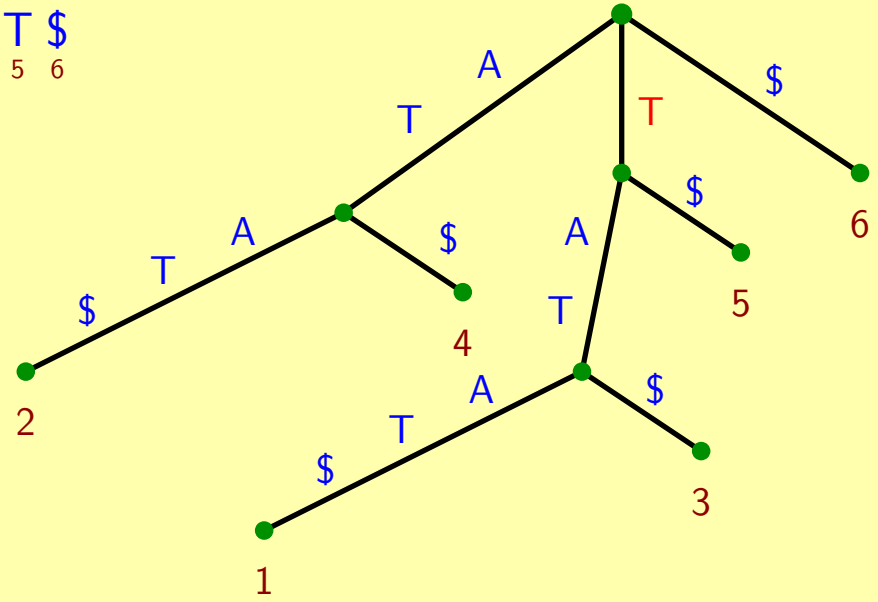
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

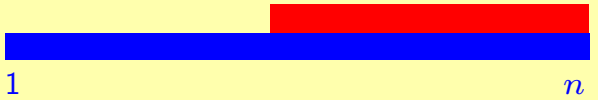
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

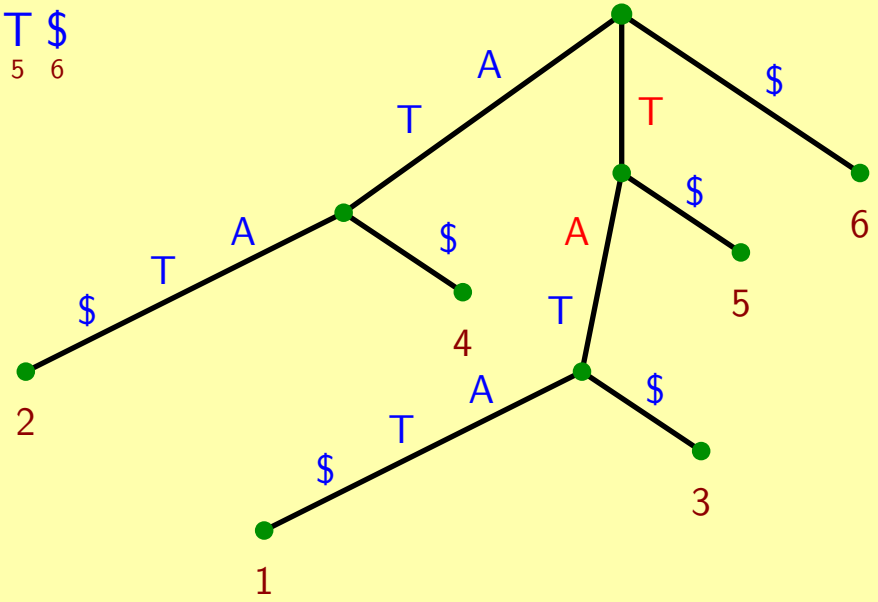
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

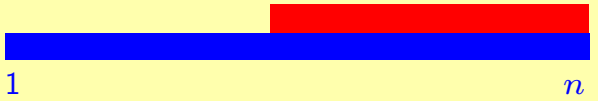
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

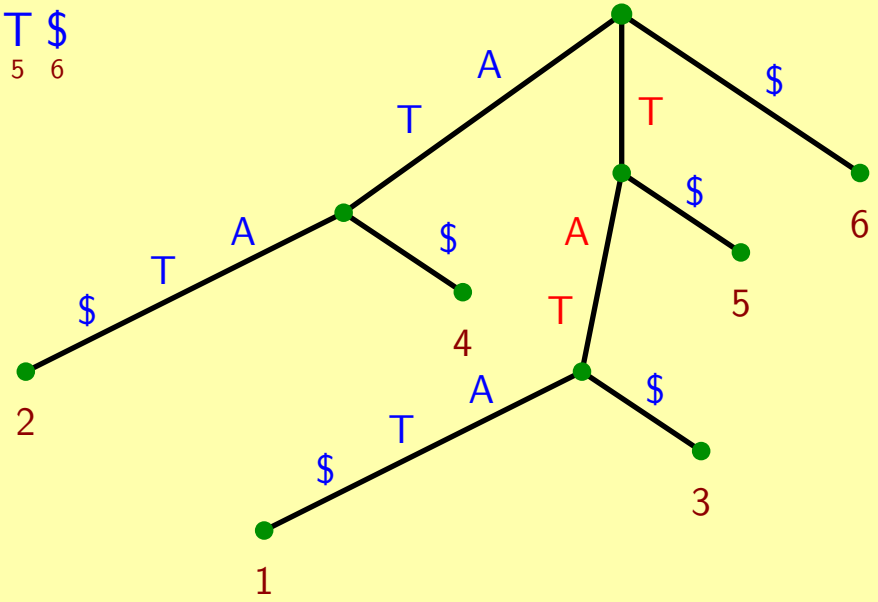
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

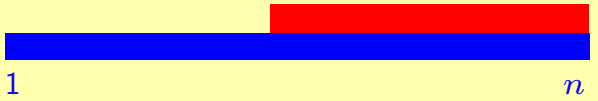
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

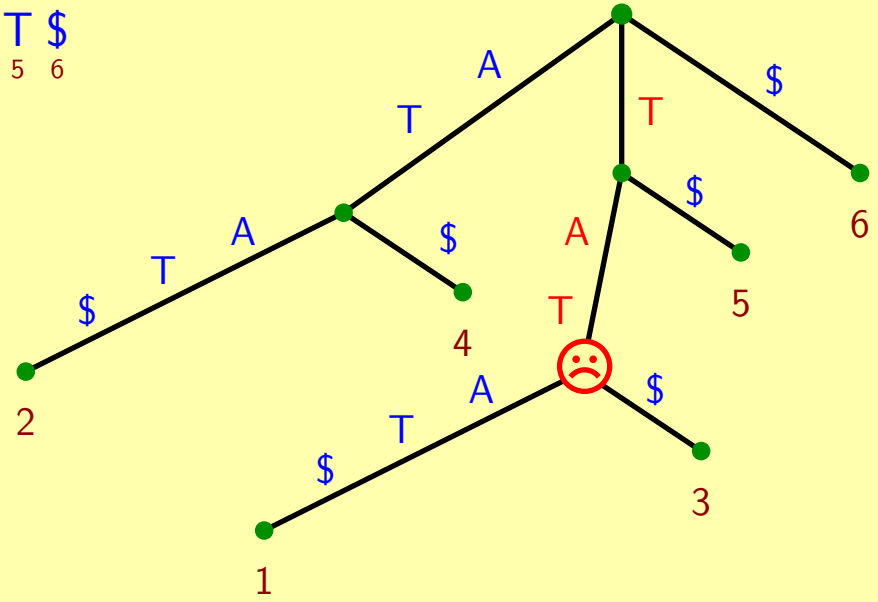
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

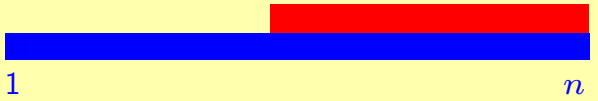
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T

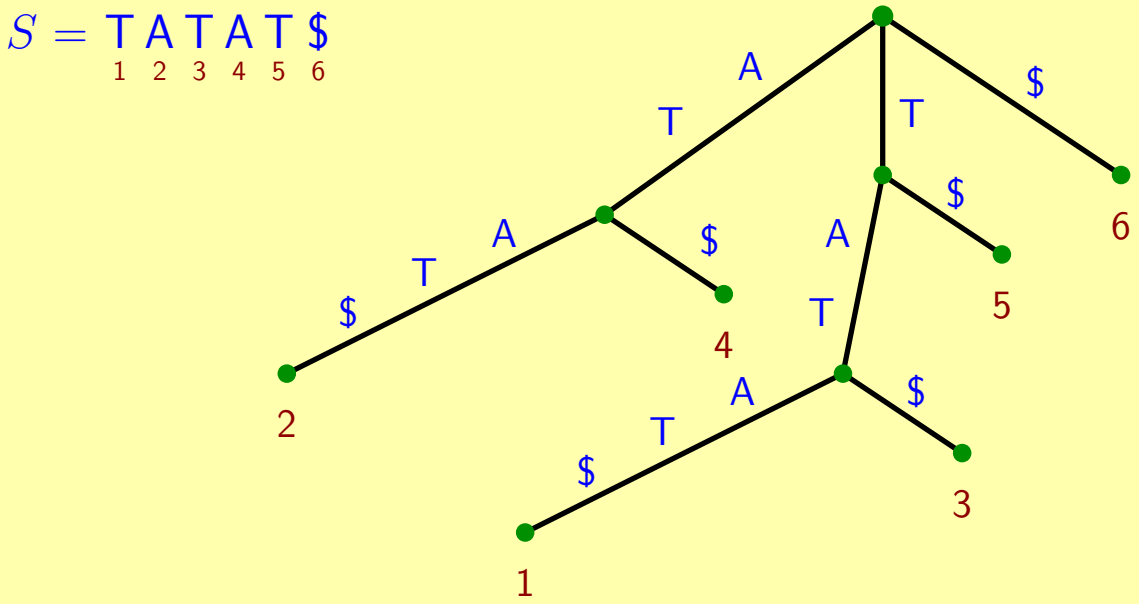


Suffix Tree: Definition

- A suffix of a string S of length n
  is a substring of S that ends at position n.

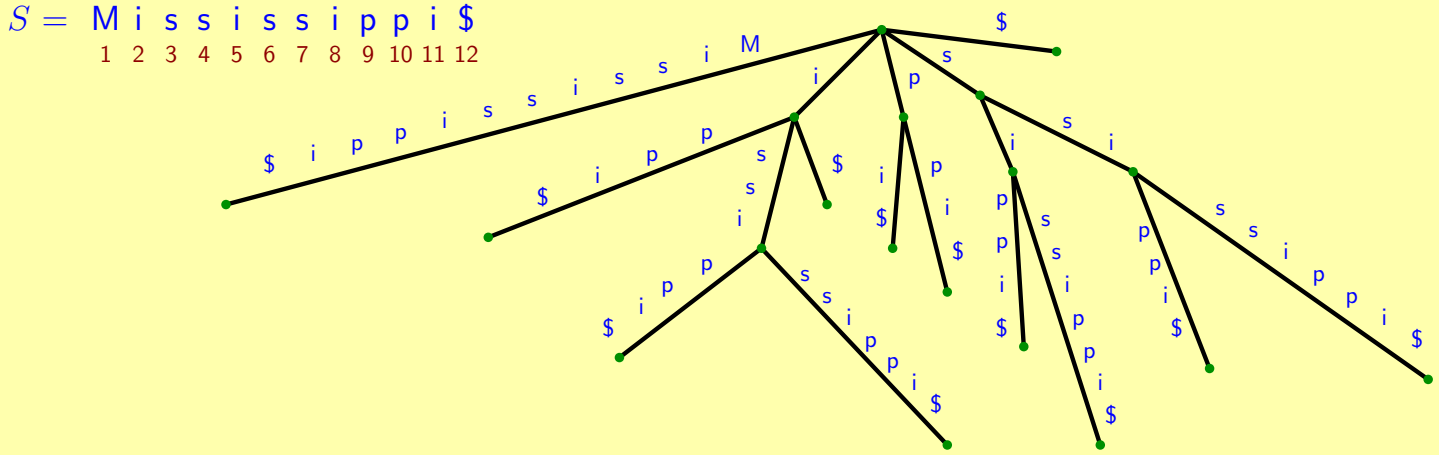


- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.



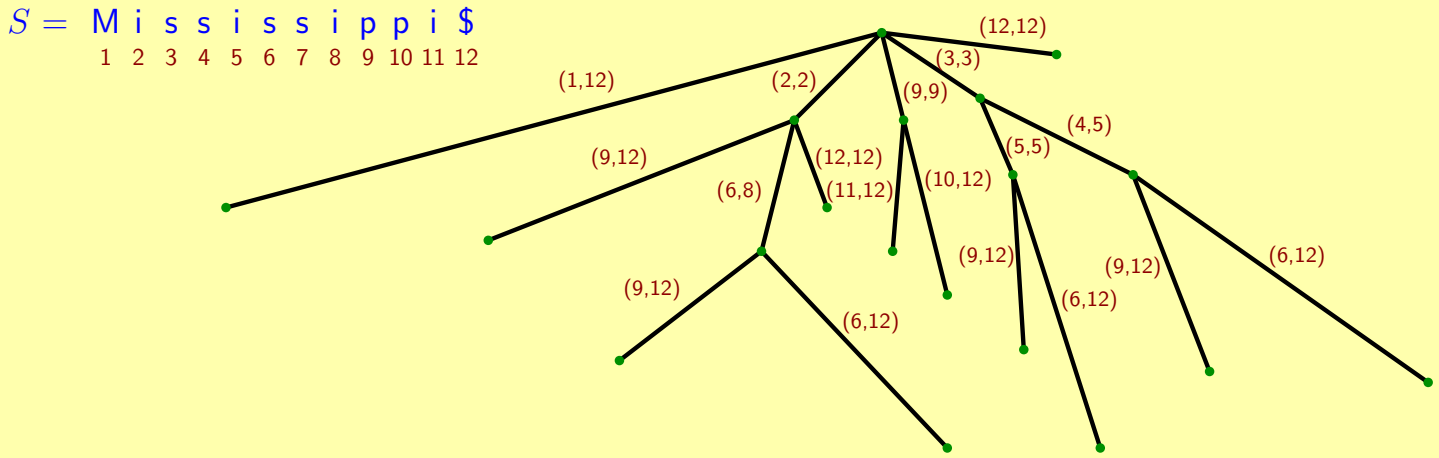
Computational aspects

- $T(S)$ requires $\mathcal{O}(n)$ space.
  Trick: Edge labels are represented by pairs of pointers into the text.



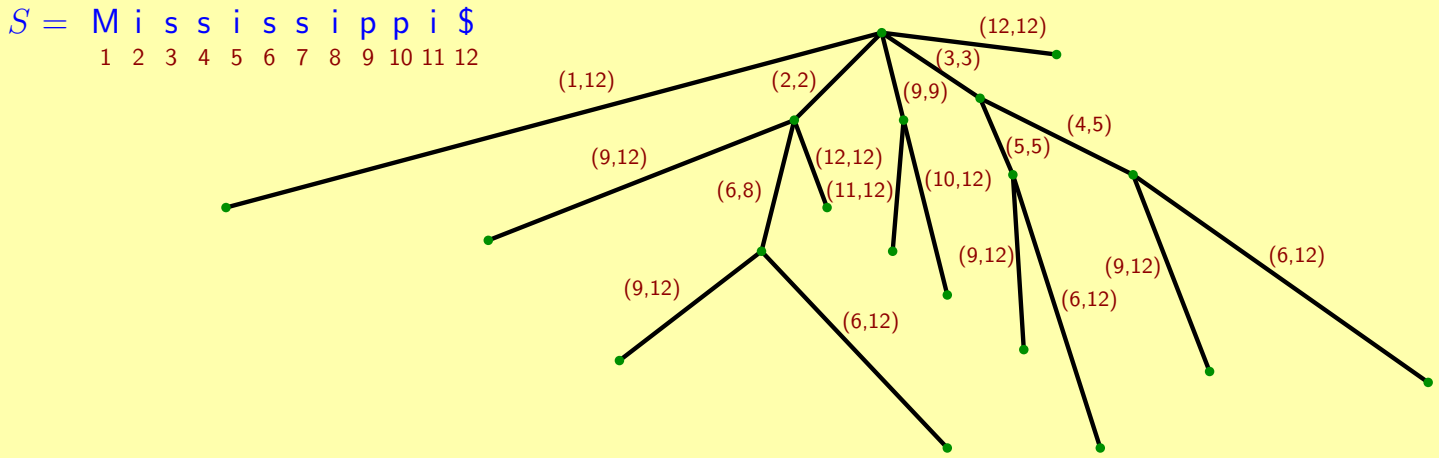
Computational aspects

- $T(S)$ requires $\mathcal{O}(n)$ space.
  Trick: Edge labels are represented by pairs of pointers into the text.



Computational aspects

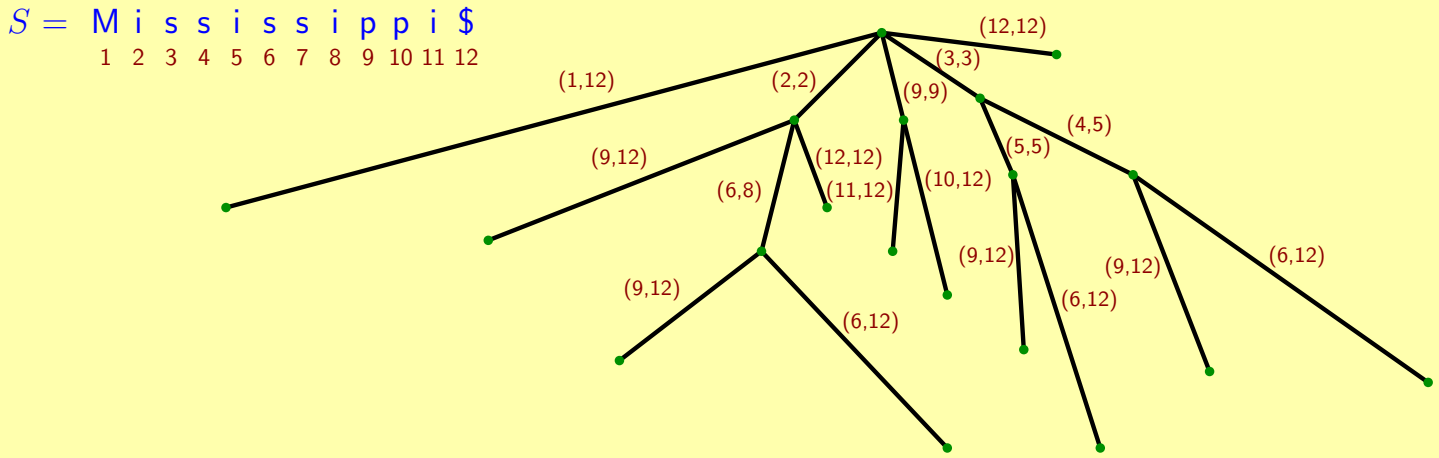
- $T(S)$ requires $\mathcal{O}(n)$ space.
  Trick: Edge labels are represented by pairs of pointers into the text.



- Actual implementations: below $12n$ bytes in worst case, $8.5n$ bytes on average

Computational aspects

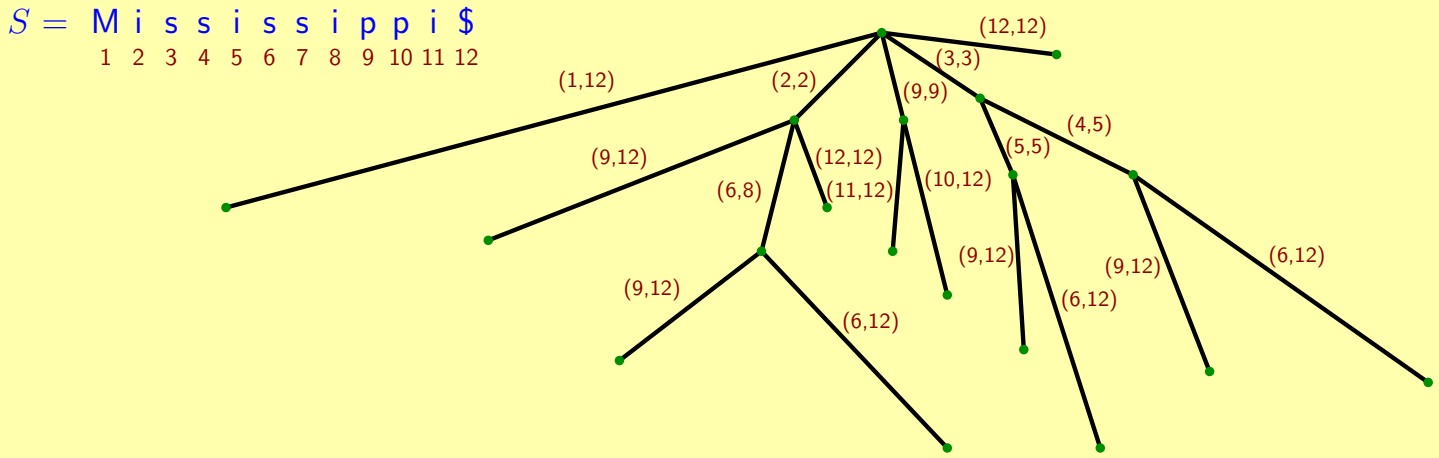
- $T(S)$ requires $\mathcal{O}(n)$ space.
  Trick: Edge labels are represented by pairs of pointers into the text.



- Actual implementations: below $12n$ bytes in worst case, $8.5n$ bytes on average
- $T(S)$ can be constructed in $\mathcal{O}(n)$ time: Weiner 1973, McCreight 1976, Ukkonen 1993.

Computational aspects

- $T(S)$ requires $\mathcal{O}(n)$ space.
  Trick: Edge labels are represented by pairs of pointers into the text.



- Actual implementations: below $12n$ bytes in worst case, $8.5n$ bytes on average
- $T(S)$ can be constructed in $\mathcal{O}(n)$ time: Weiner 1973, McCreight 1976, Ukkonen 1993.
- In practice, simpler algorithms can be much faster on average:
  Top-down construction with iterative sorting of the suffixes.

Suffix tree properties

- $T(S)$ represents exactly the substrings of S.
- $T(S)$ allows to enumerate these substrings and their locations in S in a convenient way.
- This is very useful for many pattern recognition problems, for example:
  - exact string matching as part of other applications, e.g. detecting DNA contamination
  - all-pairs suffix-prefix matching, important in fragment assembly
  - finding repeats and palindromes, tandem repeats, degenerate repeats
  - DNA primer design
  - DNA chip design
  - ...

See also:

- A. Apostolico: The myriad virtues of subword trees, 1985.
- D. Gusfield: Algorithms on strings, trees, and sequences, 1997.

Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Repeats in biosequence analysis

- DNA of eukaryotes is highly repetitive.
  - 30–50% in human genome
  - 10% introduced by retroviruses?
- Repeat regions are rapidly changing *hot spots* in evolution.
- Vast literature on repetitive structures and their hypothesized functional and evolutionary roles: ALUs, SINEs, LINEs, microsatellites, minisatellites, ...
- Repeats are involved in several biological mechanisms, including genetically inherited diseases.
  - e.g. Huntington's disease
- Repeats tend to confuse sequence analysis programs and hence should be masked in a preprocessing step.

$\Rightarrow$ Repeats are very important when studying genomic DNA.

Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

$\longrightarrow$ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

$\longrightarrow$ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

$\longrightarrow$ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

$\longrightarrow$ *k-mismatch* repeat if there are k mismatches between $S[i_1, j_1]$ and $S[i_2, j_2]$

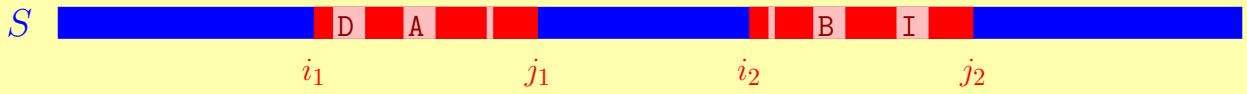
Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

$\longrightarrow$ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

$\longrightarrow$ *k-mismatch* repeat if there are k mismatches between $S[i_1, j_1]$ and $S[i_2, j_2]$

$\longrightarrow$ *k-differences* repeat if there are k differences (mismatches, insertions, deletions) between $S[i_1, j_1]$ and $S[i_2, j_2]$



Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

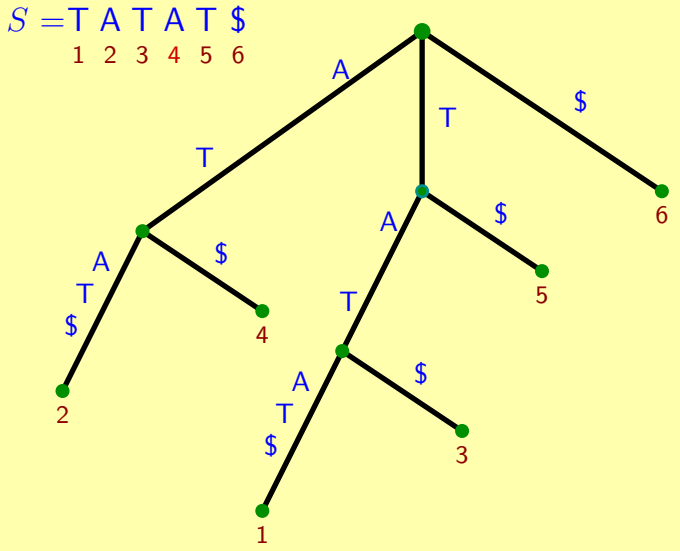
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to internal locations in $T(S)$.
- leaf numbers tell us positions where substrings occur.



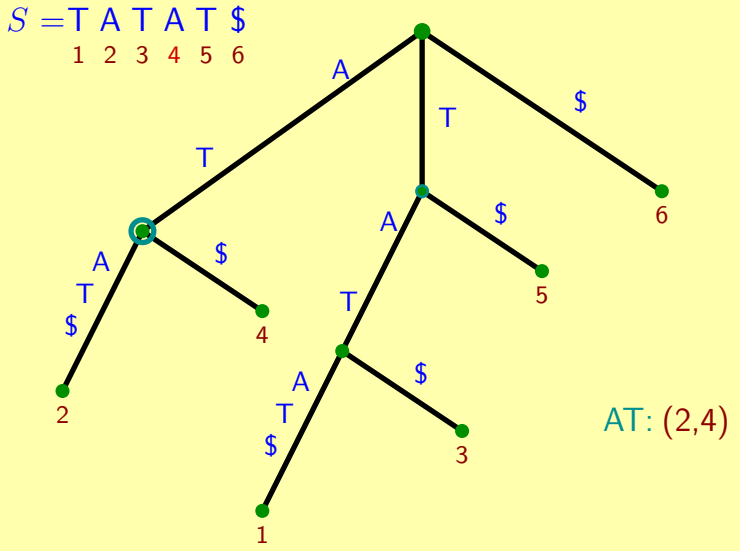
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to internal locations in $T(S)$.
- leaf numbers tell us positions where substrings occur.



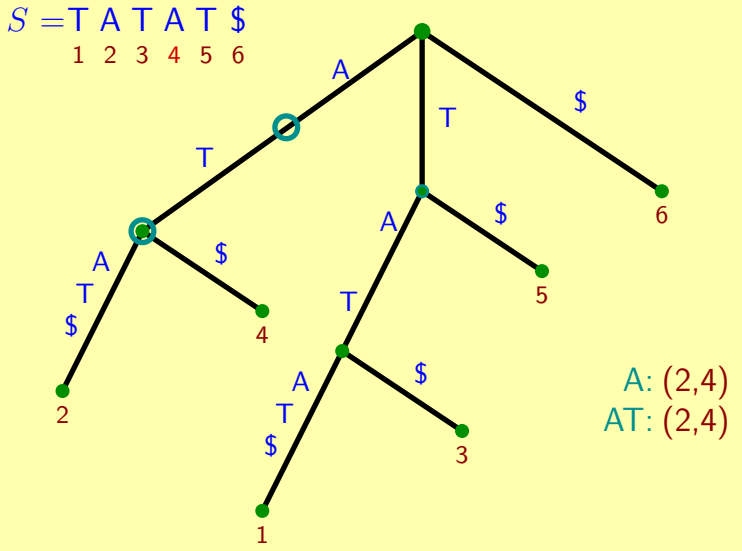
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to internal locations in $T(S)$.
- leaf numbers tell us positions where substrings occur.



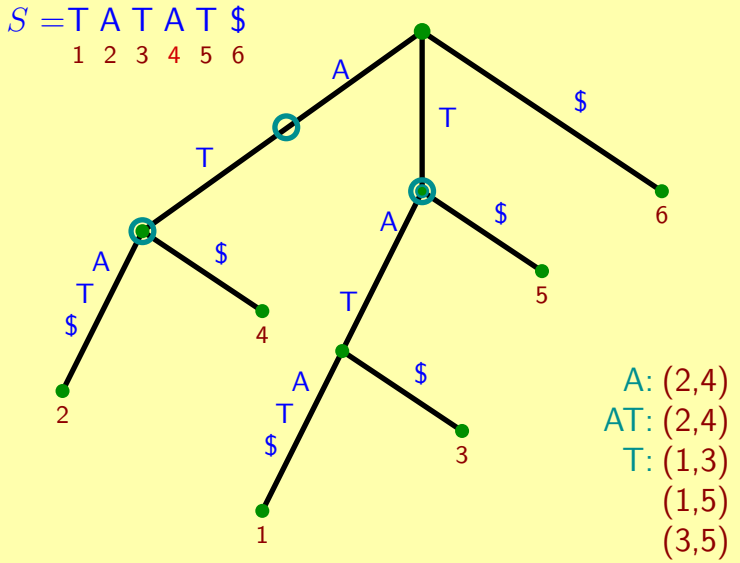
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to internal locations in $T(S)$.
- leaf numbers tell us positions where substrings occur.



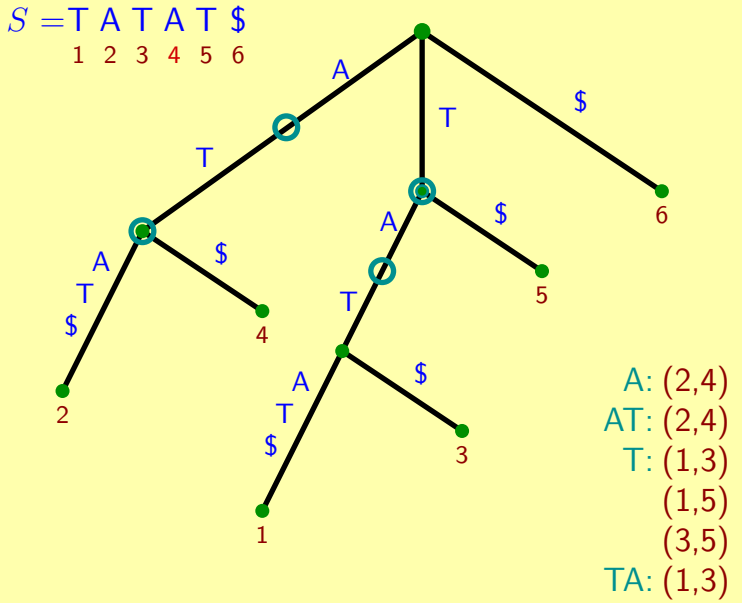
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to internal locations in $T(S)$.
- leaf numbers tell us positions where substrings occur.



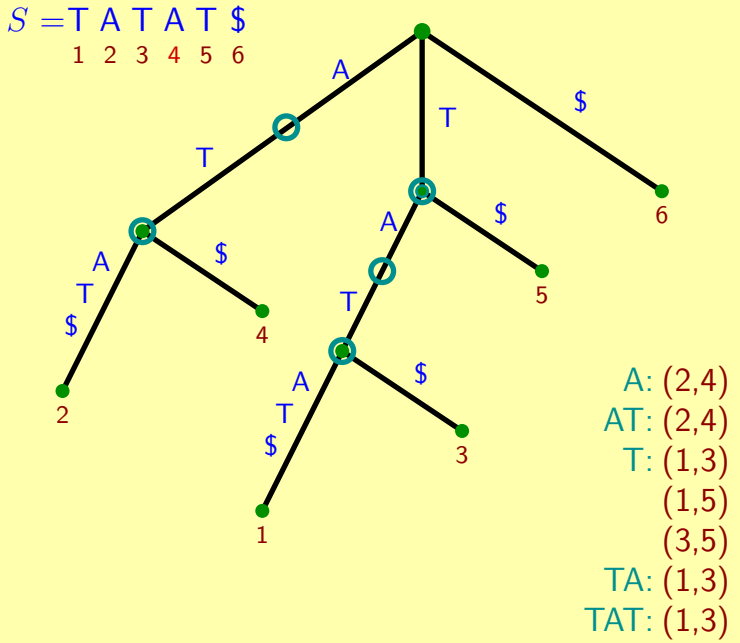
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to internal locations in $T(S)$.
- leaf numbers tell us positions where substrings occur.



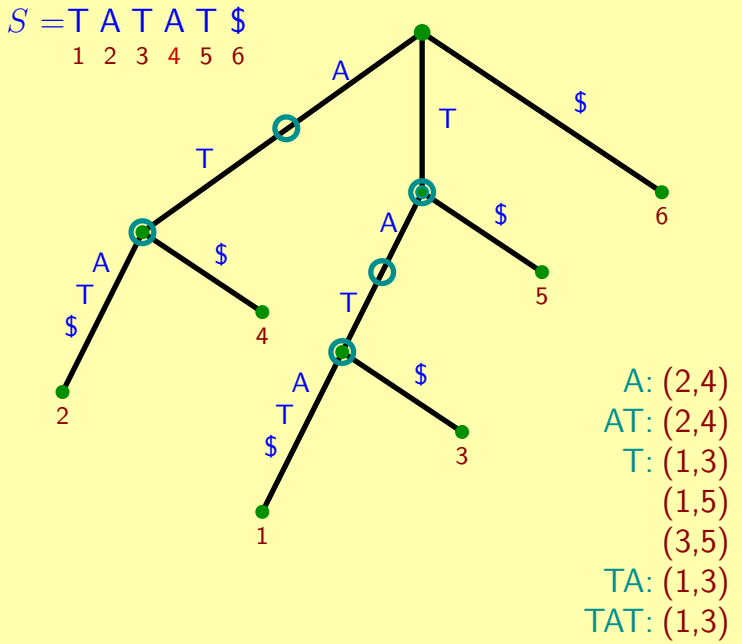
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to internal locations in $T(S)$.
- leaf numbers tell us positions where substrings occur.

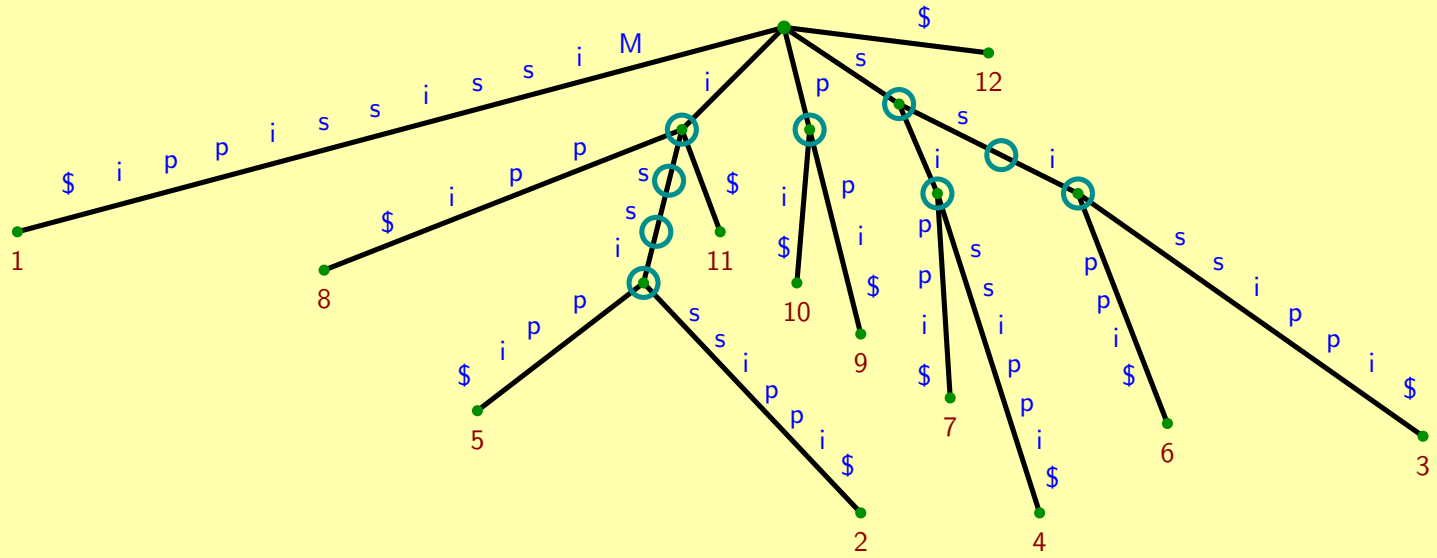


Analysis: $\mathcal{O}(n + |\text{output}|)$ time, $\mathcal{O}(n)$ space

A larger example

$S =$ M i s s i s s i p p i $

1 2 3 4 5 6 7 8 9 10 11 12



i: (8,5)	is: (5,2)	p: (10,9)	s: (7,4)	si: (7,4)
(8,2)			(7,6)	
(8,11)	iss: (5,2)		(7,3)	ss: (6,3)
(5,2)			(4,6)	
(5,11)	issi: (5,2)		(4,3)	ssi: (6,3)
(2,11)			(6,3)	

Finding *maximal* exact repeats

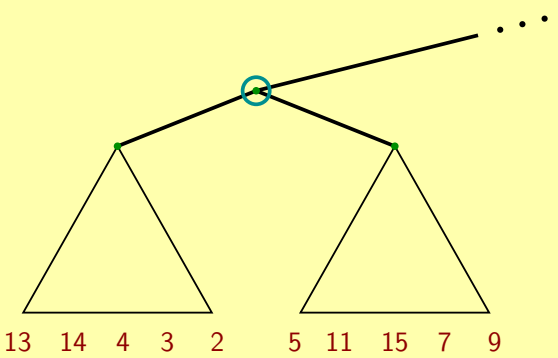
Finding *maximal* exact repeats

Finding *maximal* exact repeats

Finding *maximal* exact repeats

Idea: (see e.g. Gusfield, 1997)

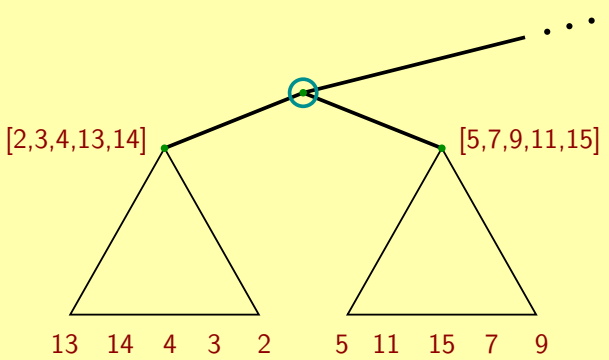
- For right-maximality ($X \neq Y$)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees



Finding *maximal* exact repeats

Idea: (see e.g. Gusfield, 1997)

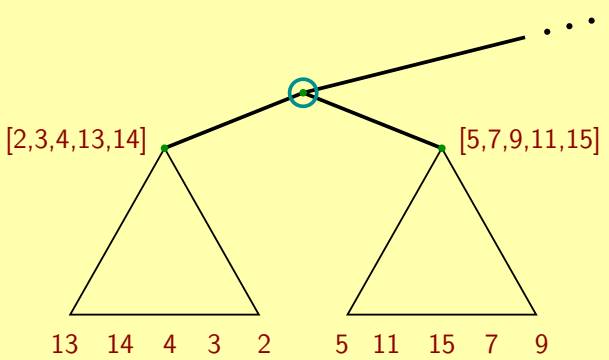
- For right-maximality (X $\neq$ Y)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees (or from different leaf-lists)



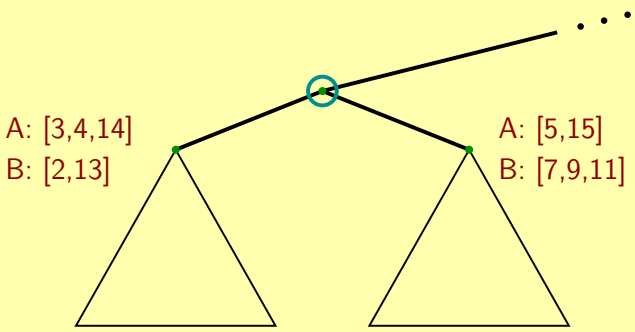
Finding *maximal* exact repeats

Idea: (see e.g. Gusfield, 1997)

- For right-maximality (X ≠ Y)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees (or from different leaf-lists)



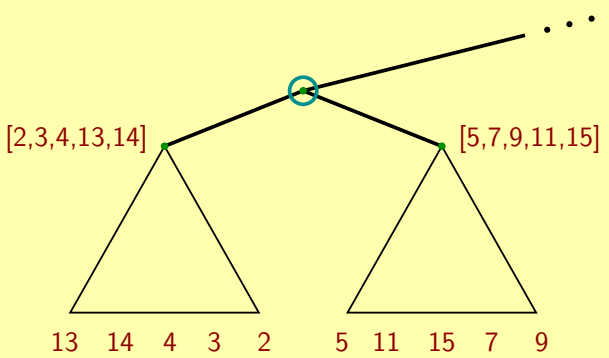
- For left-maximality (A ≠ B)
  - keep lists for the different left-characters
  - report only pairs from different lists



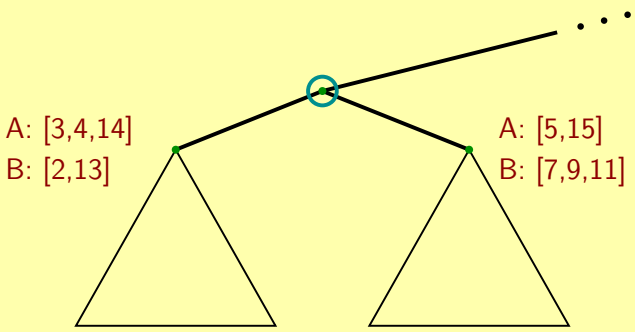
Finding *maximal* exact repeats

Idea: (see e.g. Gusfield, 1997)

- For right-maximality (X ≠ Y)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees (or from different leaf-lists)



- For left-maximality (A ≠ B)
  - keep lists for the different left-characters
  - report only pairs from different lists



Analysis: $\mathcal{O}(n + |\text{output}|)$ time, $\mathcal{O}(n)$ space

Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Tandem repeats: Definitions

- tandem repeat (square)

$$w = \boxed{\alpha}\boxed{\alpha} \quad \alpha \in \Sigma^+$$

Tandem repeats: Definitions

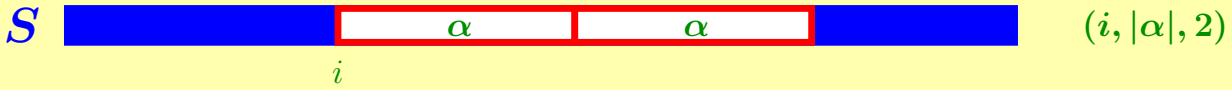
- tandem repeat (square)

- occurrence of a tandem repeat

Tandem repeats: Definitions

- tandem repeat (square)

- occurrence of a tandem repeat

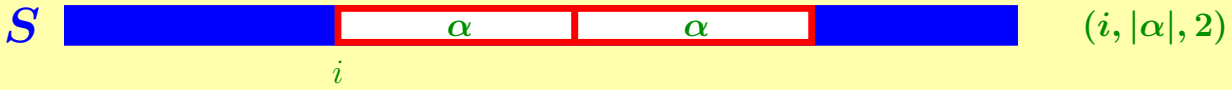


- a string w is primitive if and only if $w = u^k$ implies $k = 1$

Tandem repeats: Definitions

- tandem repeat (square)

- occurrence of a tandem repeat



- a string w is primitive if and only if $w = u^k$ implies $k = 1$
- a tandem repeat $\alpha\alpha$ is primitive if and only if α is primitive

Finding tandem repeats: Overview

A. Find all occurrences of tandem repeats in a string.

- Main & Lorentz 1979/1984: $\mathcal{O}(n \log n + |\text{output}|)$ time
- Landau & Schmidt 1993: $\mathcal{O}(n \log n + |\text{output}|)$ time

B. Find all occurrences of *primitive* tandem repeats in a string.

- Crochemore 1980/1981: $\mathcal{O}(n \log n)$ time
- Apostolico & Preparata 1983: $\mathcal{O}(n \log n)$ time
- Kolpakov & Kucherov 1998/1999: $\mathcal{O}(n + |\text{output}|)$ time

C. Find all occurrences of primitive tandem *arrays* in a string.

- Stoye & Gusfield 1998/2002: $\mathcal{O}(n \log n + |\text{output}|)$ time

Here:

Finding and representing all tandem repeats in a string in linear time.

(joint work with Dan Gusfield, to appear in JCSS)

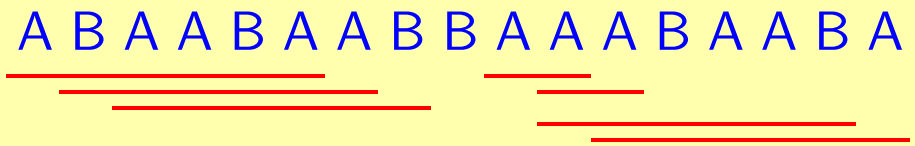
Runs of tandem repeats

There can be

- $\mathcal{O}(n^2)$ occurrences of tandem repeats;
- $\mathcal{O}(n \log n)$ occurrences of primitive tandem repeats.

$\Rightarrow$ Any efficient algorithm to enumerate all occurrences of tandem repeats in a string will depend on the output size.

Definition: A series of tandem repeats of the same length, occurring in S at contiguous positions, is called a run of tandem repeats.



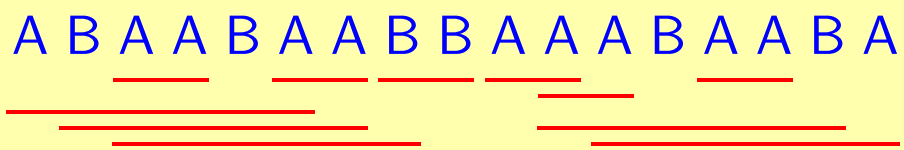
Theorem (Kolpakov & Kucherov 1999):
The number of runs of primitive tandem repeats is bounded by $\mathcal{O}(n)$.

Algorithm: Find runs in $\mathcal{O}(n)$ time, then list all tandem repeats in $\mathcal{O}(n + |\text{output}|)$ time.

The vocabulary of tandem repeats

Definitions:

- Two occurrences of tandem repeats (i, l) and (i', l) are of the same type if and only if $S[i..i+l-1] = S[i'..i'+l-1]$.
- The vocabulary of tandem repeats $V(S)$ is the set of tandem repeat types contained in S.



$$V(S) = \{\mathsf{ABAABA}, \mathsf{BAABAA}, \mathsf{AABAAB}, \mathsf{AA}, \mathsf{BB}\}$$

Theorem (Fraenkel & Simpson 1998): $|V(S)| \leq 2n$

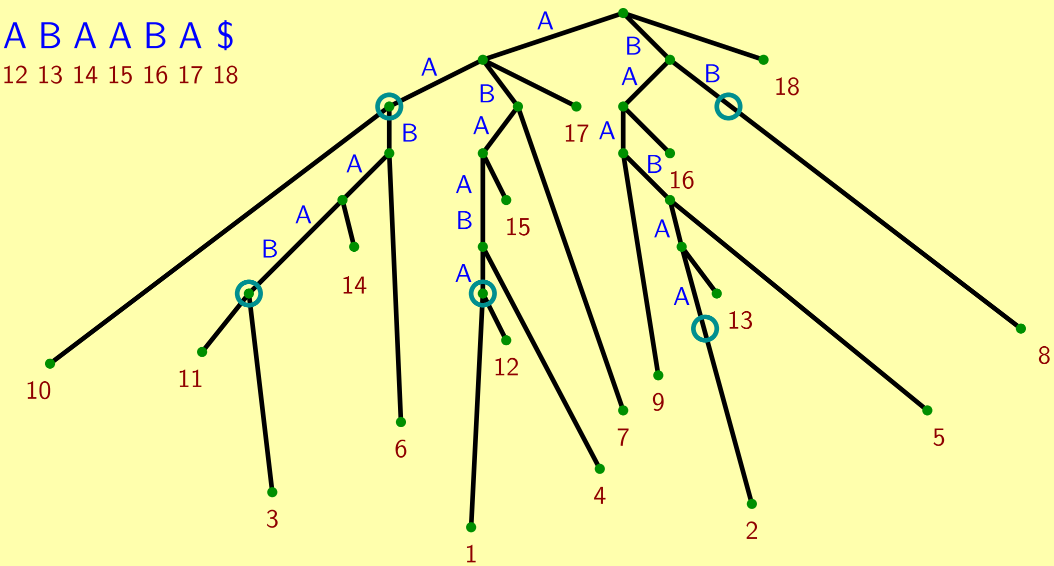
Challenge: Can we find the vocabulary of tandem repeats in $\mathcal{O}(n)$ time?

Result: Annotation of the suffix tree

Mark in $T(S)$ the endpoints of all tandem repeats in $\mathcal{O}(n)$ time and space.

$S =$ A B A A B A A B B A A A B A A B A \$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18



Extensions:

- Find all occurrences of tandem repeats in $\mathcal{O}(n + |\text{output}|)$ time.
- Find the number, shortest, longest, ... tandem repeat in $\mathcal{O}(n)$ time.
- Find the vocabulary of primitive tandem repeats in $\mathcal{O}(n + |\text{output}|)$ time.
- Find (primitive) tandem arrays in optimal time.

Outline of the algorithm

Phase I

Using the Lempel-Ziv decomposition of S in combination with longest common extension queries, find a subset of the occurrences of tandem repeats, a leftmost covering set.

Phase II

Find the endpoints in the suffix tree of S for some of the tandem repeats in the leftmost covering set.

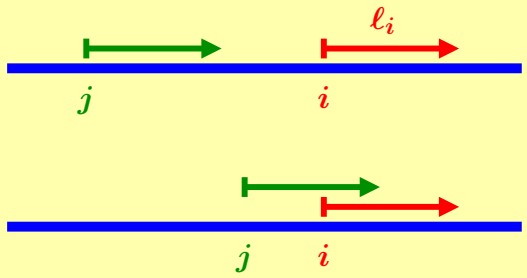
Phase III

Traverse parts of the suffix tree from the endpoints found in Phase II, to obtain the complete vocabulary of tandem repeats.

The Lempel-Ziv decomposition

Definitions:

- For each position i of S, let ℓ_i denote the length of the longest prefix of $S[i..n]$ that also occurs as a substring of S starting at some position $j < i$.



- The Lempel-Ziv (LZ) decomposition of S is the list of indices $i_1, i_2, \ldots, i_k$, defined inductively by $i_1 = 1$ and $i_{B+1} = i_B + \max(1, \ell_{i_B})$ for $i_B \leq n$.

A	B	A	A B A A B	B A A	A B A A B A	$
1	2	3	4 5 6 7 8	9 10 11	12 13 14 15 16 17	18
1	2	3	4	5	6	7

$\rightarrow$ can be computed in $\mathcal{O}(n)$ time, e.g. using the suffix tree of S

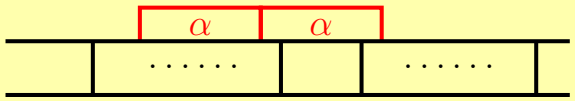
LZ decomposition and tandem repeats

(Crochemore 1981, Main 1989)

Lemma 1:

The right half of any tandem repeat occurrence must touch at most two blocks of the LZ decomposition.

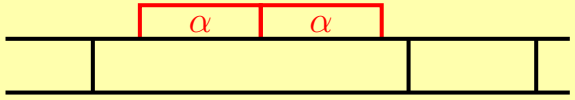
Not possible:



Lemma 2:

The leftmost occurrence of any tandem repeat must touch at least two blocks.

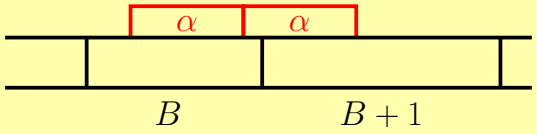
Not possible:



Theorem:

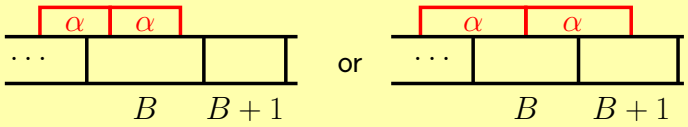
If the leftmost occurrence of a tandem repeat $\alpha\alpha$ has its center in some block B, then either

(1) $\alpha\alpha$ has its left end in block B and its right end in block $B+1$;



or

(2) the left end of $\alpha\alpha$ extends into block $B-1$ and possibly further left.

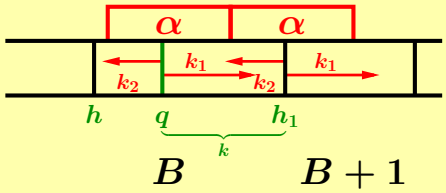


Phase I

Process every block B of the LZ decomposition by the following two algorithms.

Algorithm 1a

For $k = 1..|B|$ do:



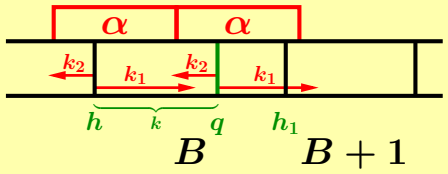
$\left.\begin{matrix} k_1 \\ k_2 \end{matrix}\right\}$ = l.c.e. from q and h_1 in the $\left\{\begin{matrix} \text{forward} \\ \text{backward} \end{matrix}\right\}$ direction

longest common extension

If $k_1 + k_2 \geq k$, then output $(q - k_2, 2k)$.

Algorithm 1b

For $k = 1..|B| + |B + 1|$ do:



$\left.\begin{matrix} k_1 \\ k_2 \end{matrix}\right\}$ = l.c.e. from h and q in the $\left\{\begin{matrix} \text{forward} \\ \text{backward} \end{matrix}\right\}$ direction

If $k_1 + k_2 \geq k$, then output $(h - k_2, 2k)$.

Analysis:

Algorithm 1 outputs a leftmost covering set in $\mathcal{O}(n)$ time and space.

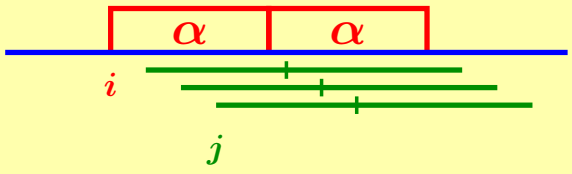
Leftmost covering sets

Idea:

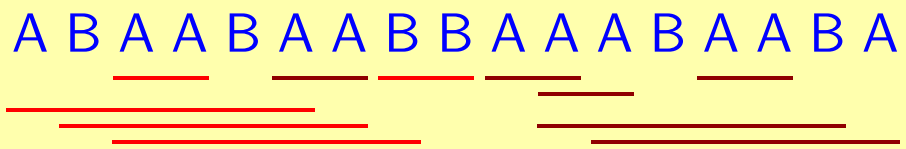
Find a subset of tandem repeats, such that by successive right-rotations all the tandem repeat types can be obtained.

Definitions:

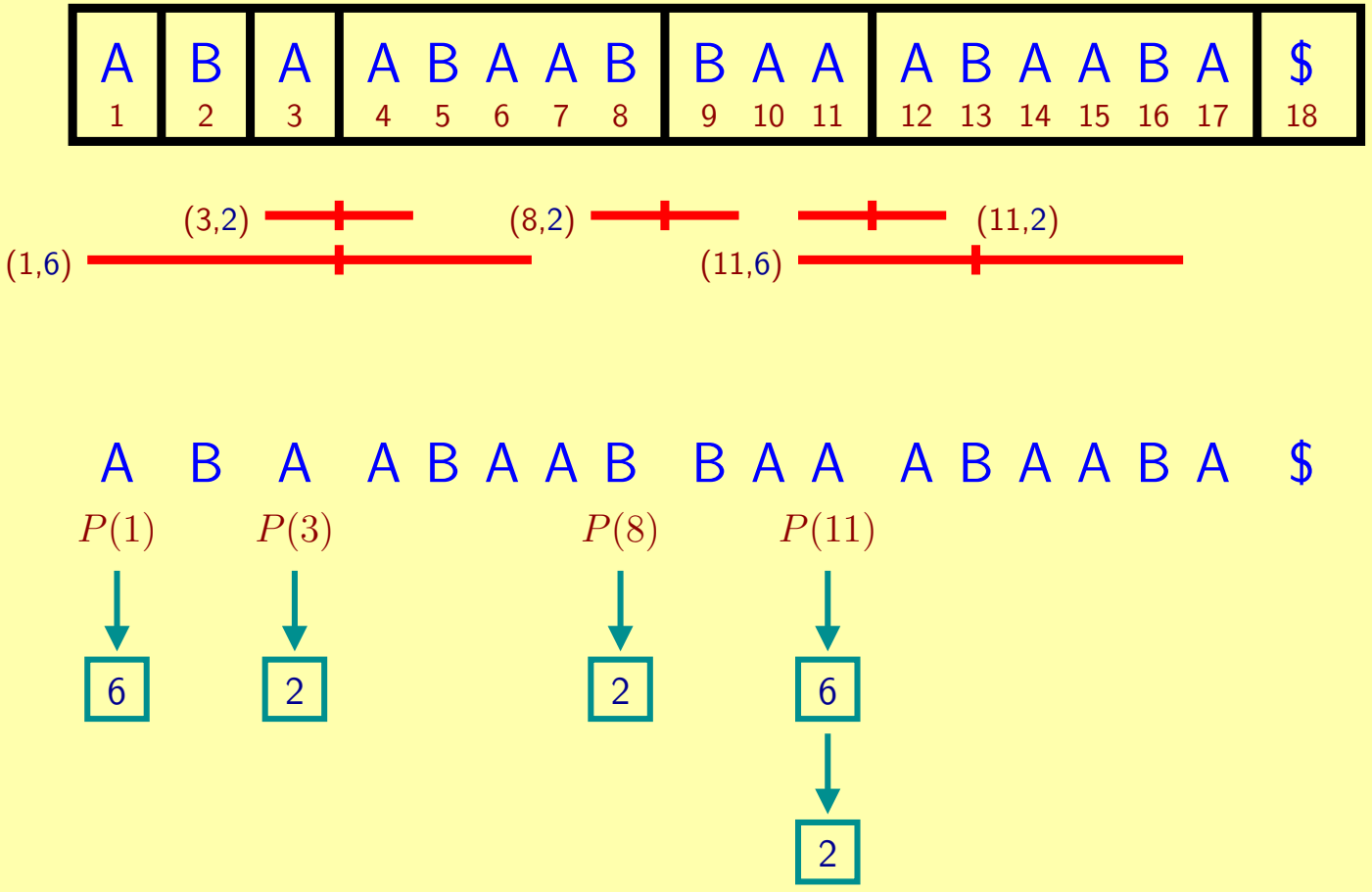
- A tandem repeat occurrence (i, ℓ) covers another occurrence (j, ℓ) if and only if there is a run of tandem repeats that starts at i and contains j.



- A set of tandem repeat occurrences Q is a leftmost covering set if the leftmost occurrence of each tandem repeat in $V(S)$ is covered by some occurrence in Q.



Picture after Phase I



Phase II

Algorithm

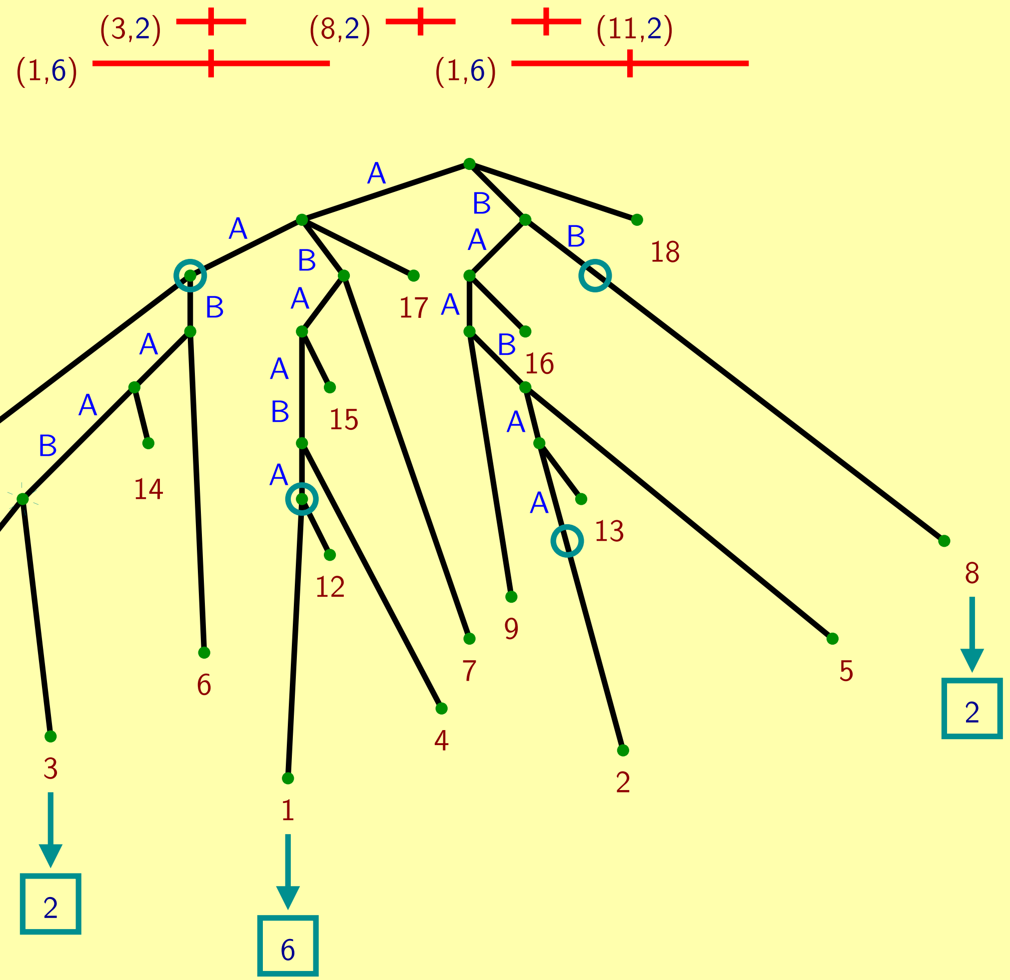
1. Attach the list $P(i)$ to the edge ending at leaf i.
2. Traverse $T(S)$ bottom-up.
   - When traversing an edge e,
     - ◇ test the head of e's list for tandem repeats ending in e;
     - ◇ record these in e, and remove the entries from the list.
       (Note: ≤ 2 tandem repeats per edge!)
   - At a node, attach the list originating from the leaf with the smallest index to the edge leading upwards.

Analysis: $\mathcal{O}(n)$ time an space.

Picture after Phase II

$S =$ A B A A B A A B B A A A B A A B A $

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

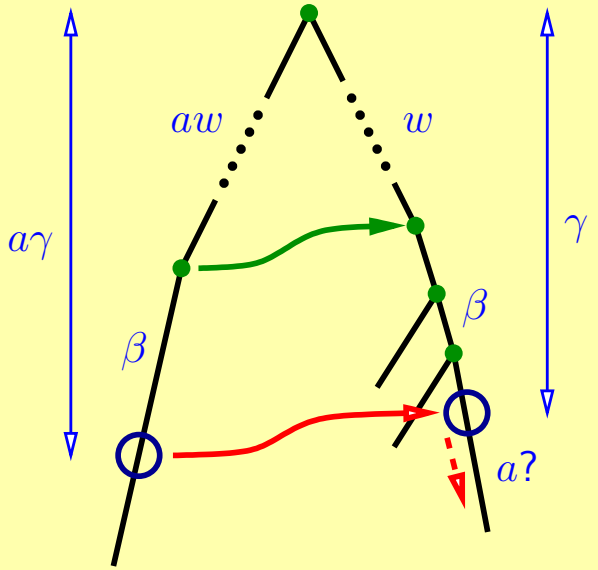


Phase III

Definition:

A suffix-link walk from the endpoint of $a\gamma$ moves to the location in $T(S)$ labeled with the string γ.

If there is a continuation a, then the walk is called successful, otherwise unsuccessful.



Algorithm:

1. From each location of tandem repeats found in Phase II, start a chain of suffix-link walks.
2. This chain ends the first time an unsuccessful walk ends, or the first time that a successful walk ends at the endpoint of a tandem repeat that has already been recorded in $T(S)$.

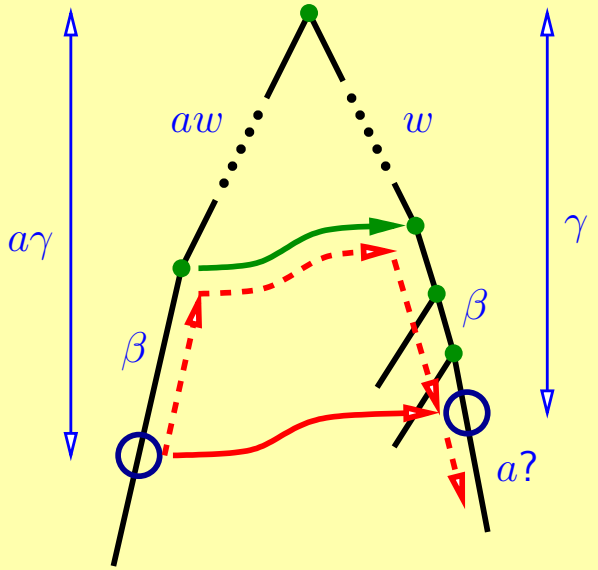
Analysis: $\mathcal{O}(n)$ time and space.

Phase III

Definition:

A suffix-link walk from the endpoint of $a\gamma$ moves to the location in $T(S)$ labeled with the string γ.

If there is a continuation a, then the walk is called successful, otherwise unsuccessful.

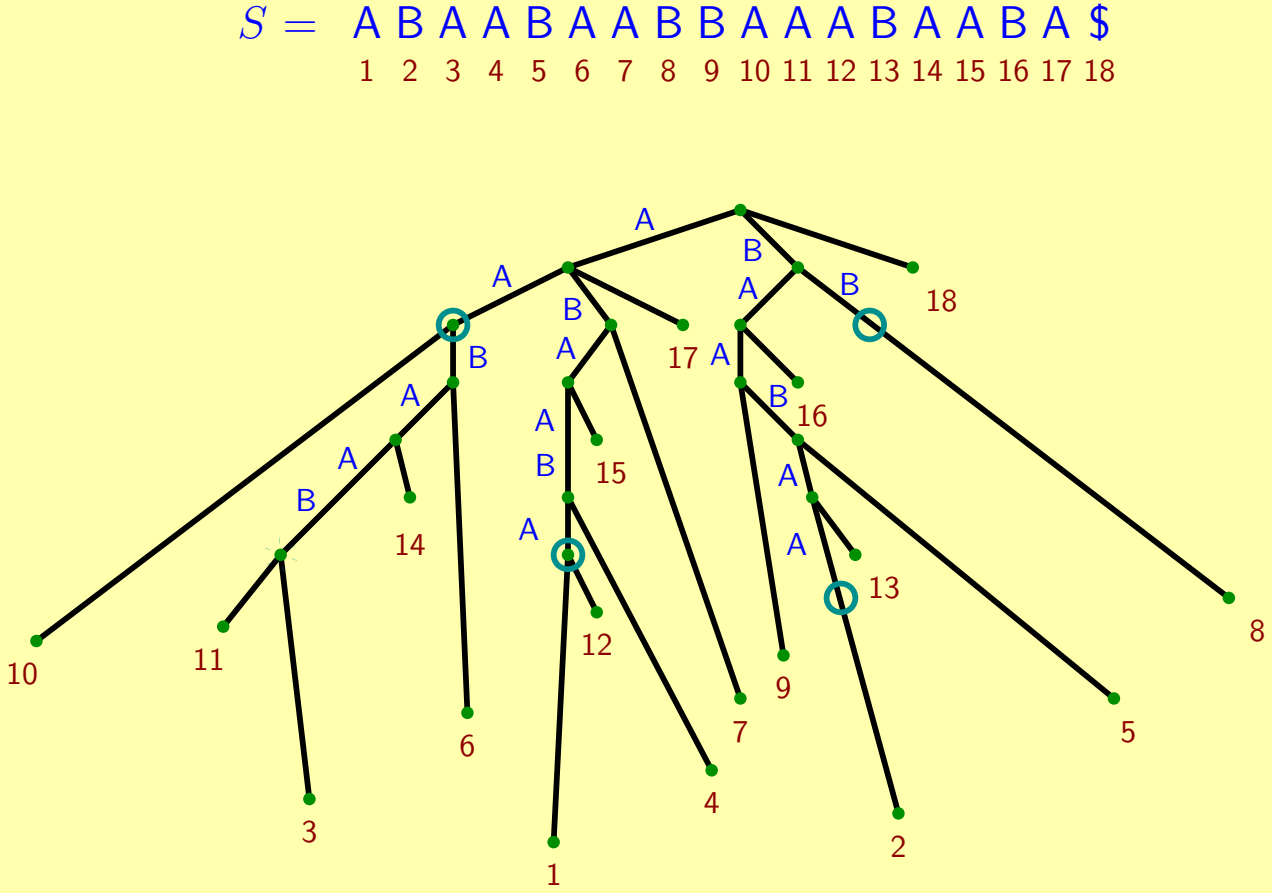


Algorithm:

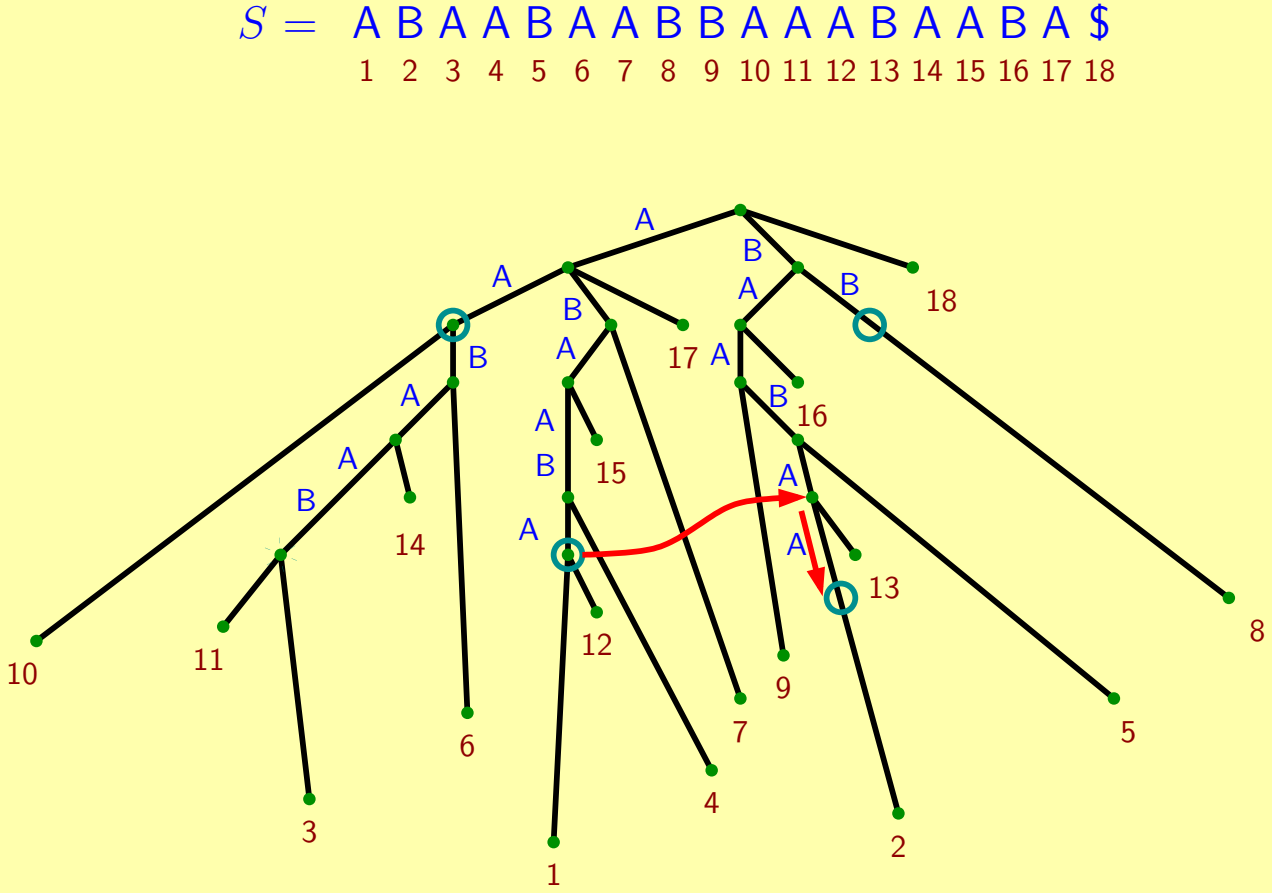
1. From each location of tandem repeats found in Phase II, start a chain of suffix-link walks.
2. This chain ends the first time an unsuccessful walk ends, or the first time that a successful walk ends at the endpoint of a tandem repeat that has already been recorded in $T(S)$.

Analysis: $\mathcal{O}(n)$ time and space.

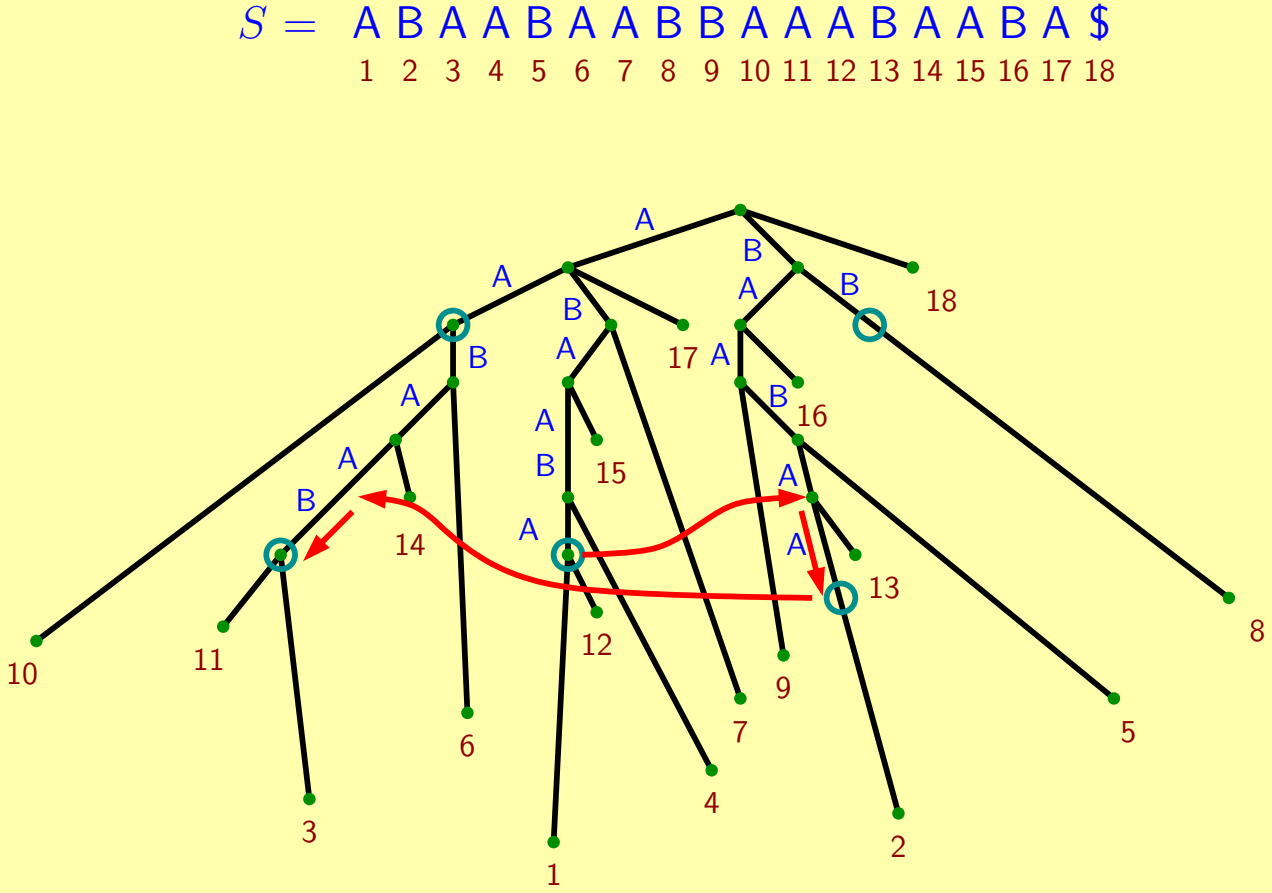
Final picture (after Phase III)



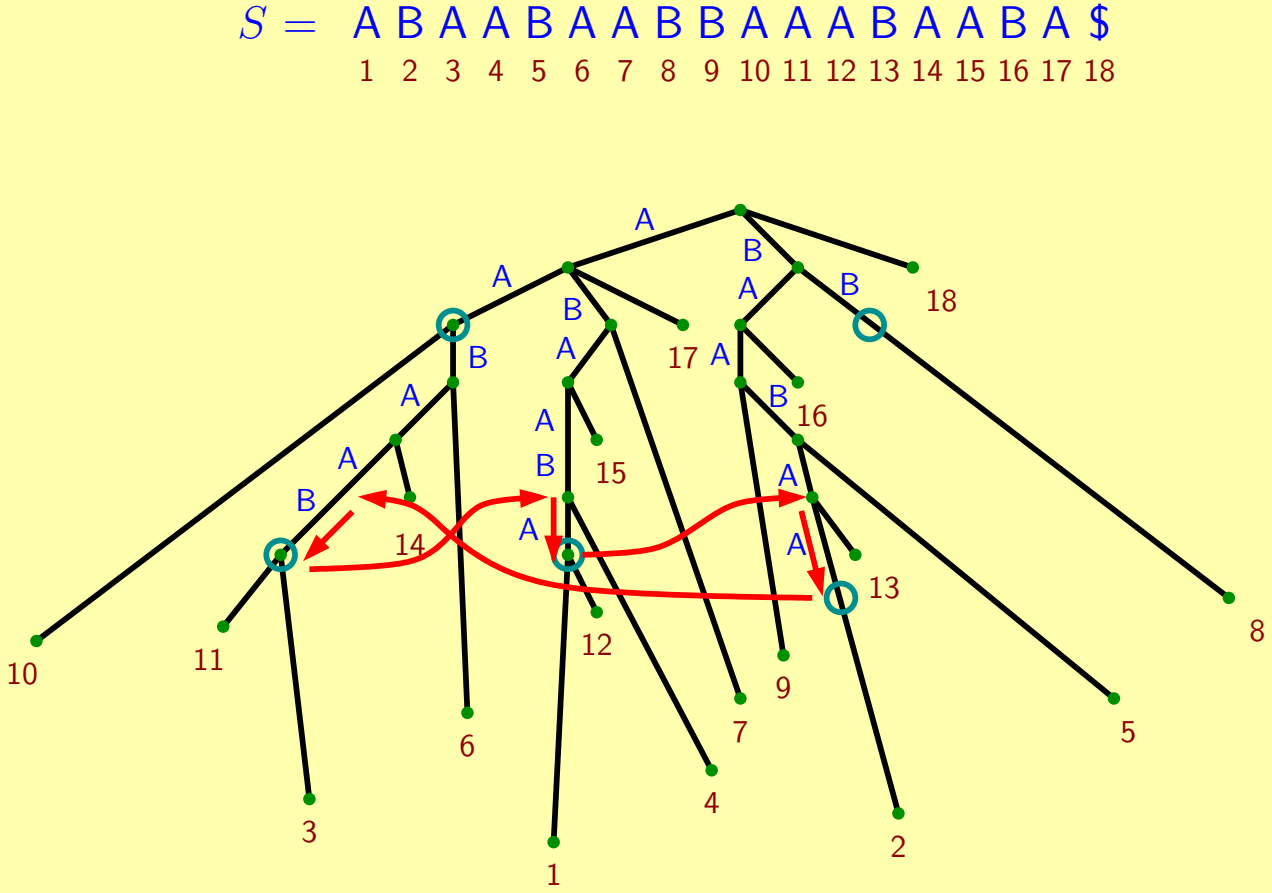
Final picture (after Phase III)



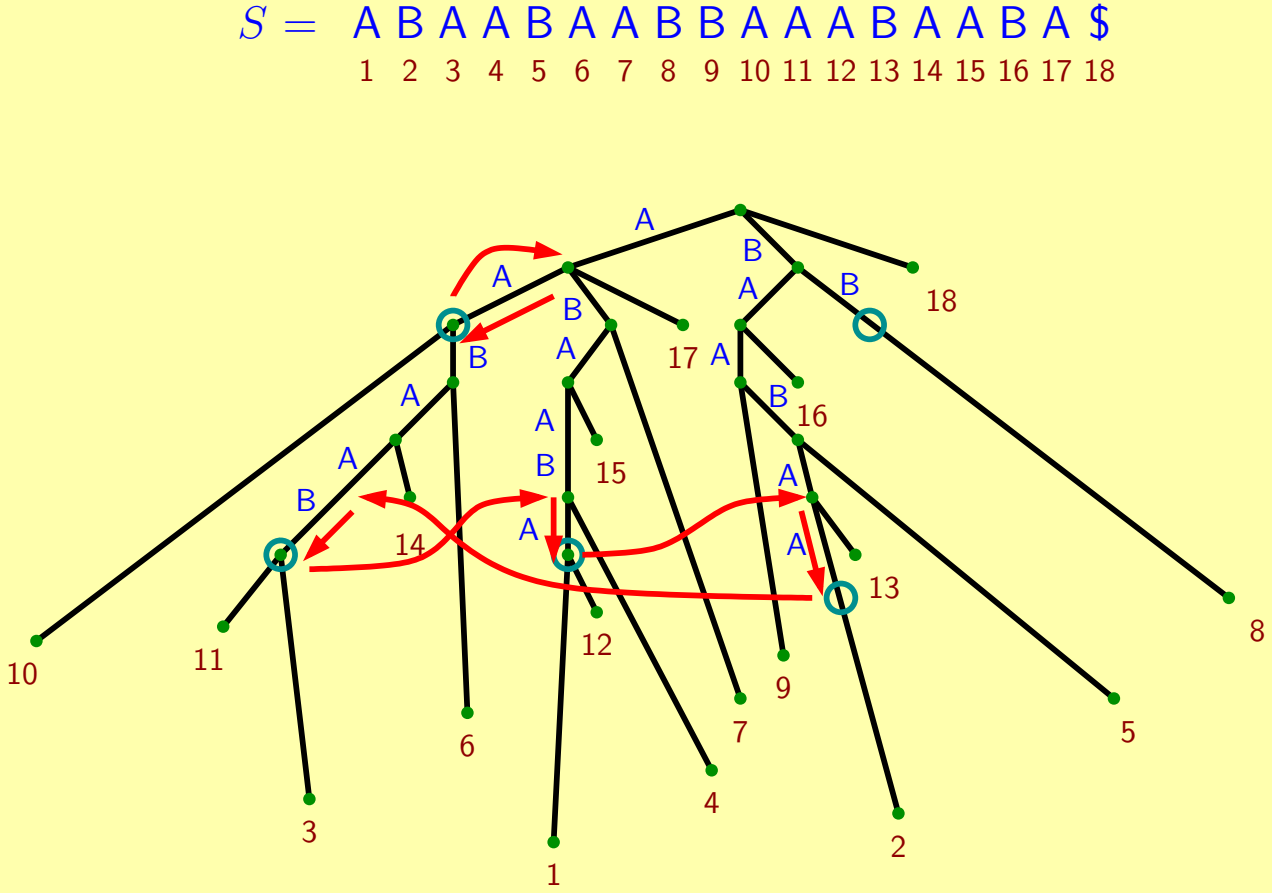
Final picture (after Phase III)



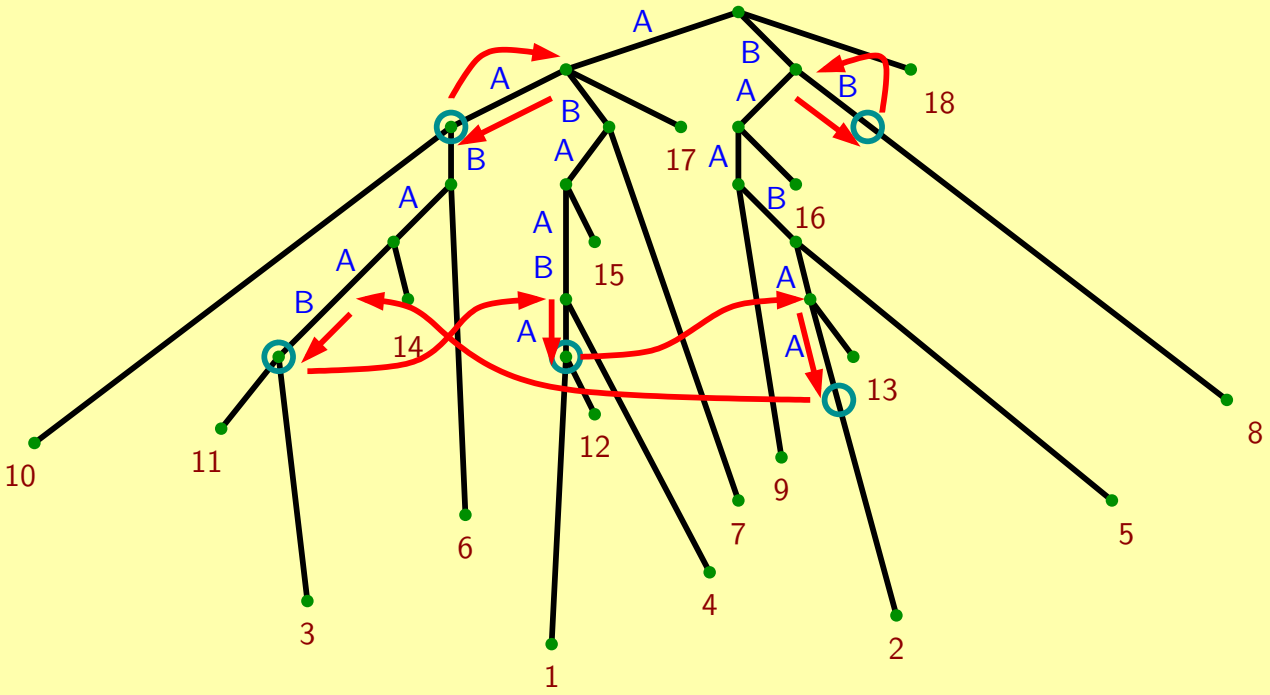
Final picture (after Phase III)



Final picture (after Phase III)



Final picture (after Phase III)



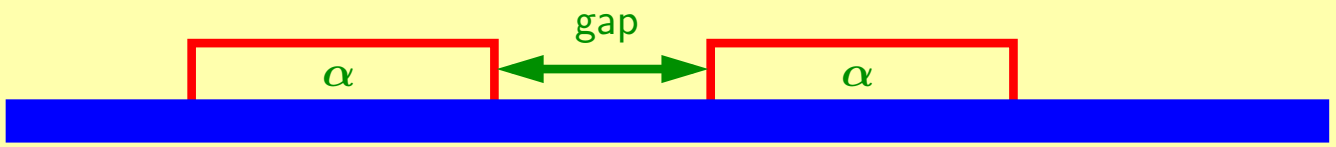
Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Repeats with bounded gap

(joint work with G. S. Brodal, R. B. Lyngø, C. N. S. Pedersen)

Sometimes one wishes to allow between the copies of a repeat a gap of (upper and/or lower) bounded size.



Idea:

- Traverse the suffix tree bottom-up.
- At each vertex v collect the leaf-list $LL'(v)$.
- Output only pairs that have the required distance.

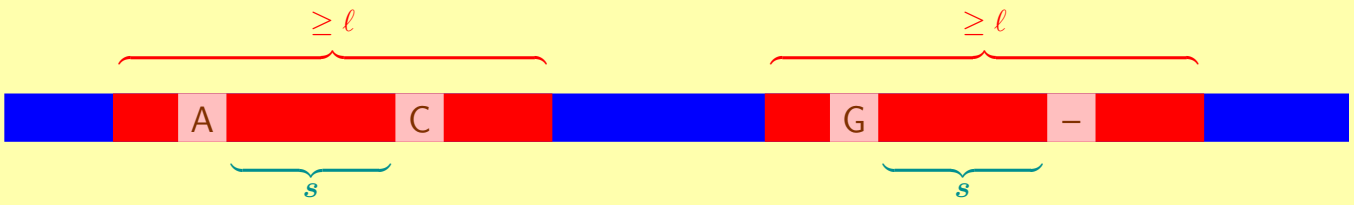
Analysis: $\mathcal{O}(n \log n + |\text{output}|)$ resp. $\mathcal{O}(n + |\text{output}|)$ time, $\mathcal{O}(n)$ space.

Finding *degenerate* repeats

(joint work with R. Giegerich, S. Kurtz, E. Ohlebusch, C. Schleiermacher)

Often, repats in genomic DNA are degenerate (k-mismatch repeats, k-differences repeats).

Idea: Minimal length ℓ, up to k errors $\longrightarrow$ filter method ("seed and extend")



$s \geq \left\lfloor \frac{\ell}{k+1} \right\rfloor$

Algorithm:

1. Search for local exact repeats (seeds).
2. Extend the seeds while allowing up to k errors.
3. If extension is long enough, output repeat.

Analysis: $\mathcal{O}(n + \zeta k^3)$ time with $E(\zeta) = O\left(n^2/4^s\right)$, s minimal seed length.

Extension for maximal k-mismatch repeats

Simple extension and length test (minimal length ℓ, up to k errors):

Extension for maximal k-mismatch repeats

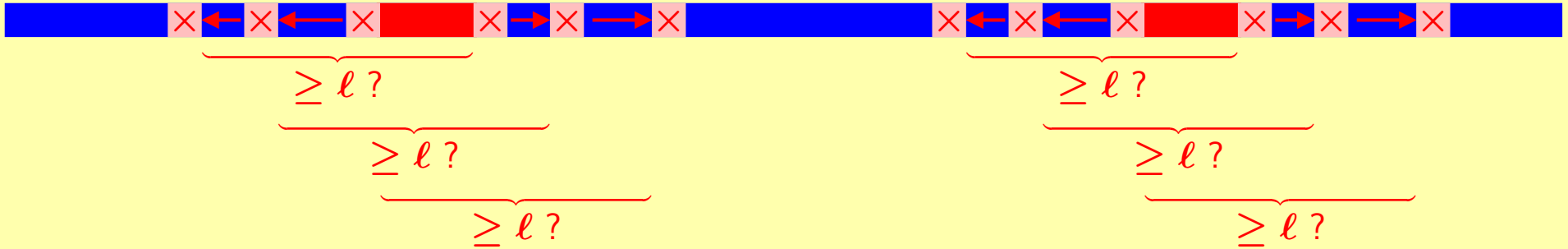
Simple extension and length test (minimal length ℓ, up to k errors):

Extension for maximal k-mismatch repeats

Simple extension and length test (minimal length ℓ, up to k errors):

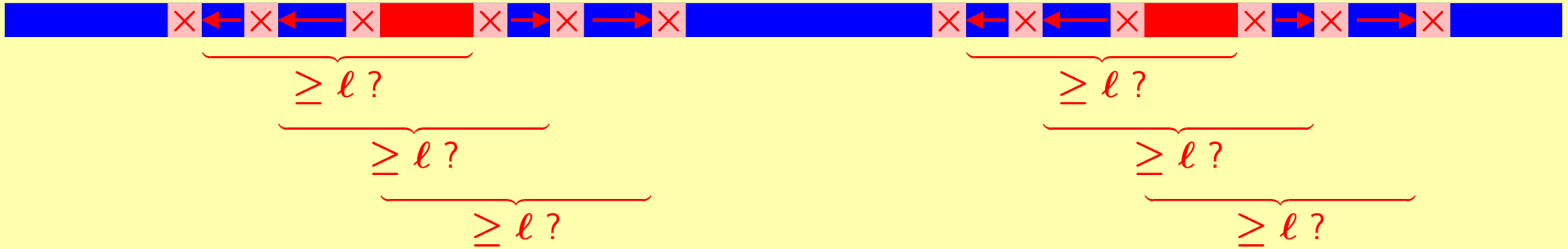
Extension for maximal k-mismatch repeats

Simple extension and length test (minimal length ℓ, up to k errors):



Extension for maximal k-mismatch repeats

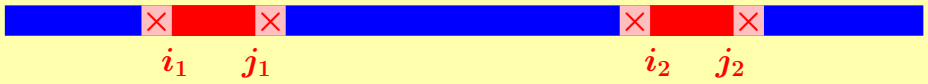
Simple extension and length test (minimal length ℓ, up to k errors):



Analysis: $\mathcal{O}(n + \zeta k)$ time with $E(\zeta) = O\left(n^2/4^s\right)$, s minimal seed length.

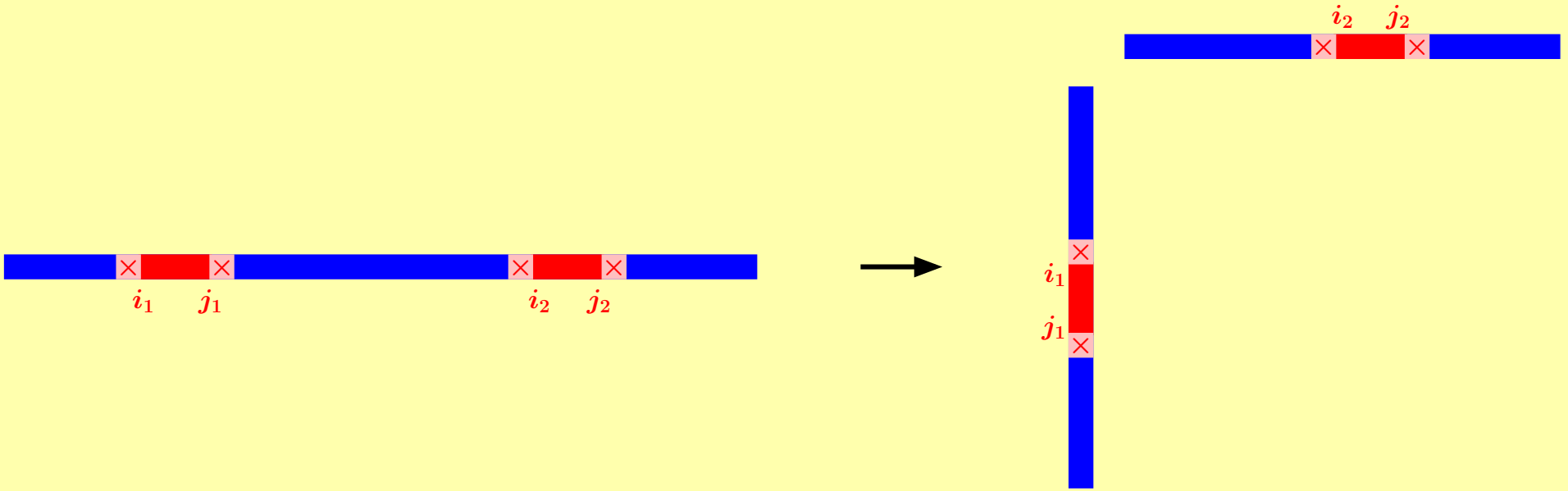
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



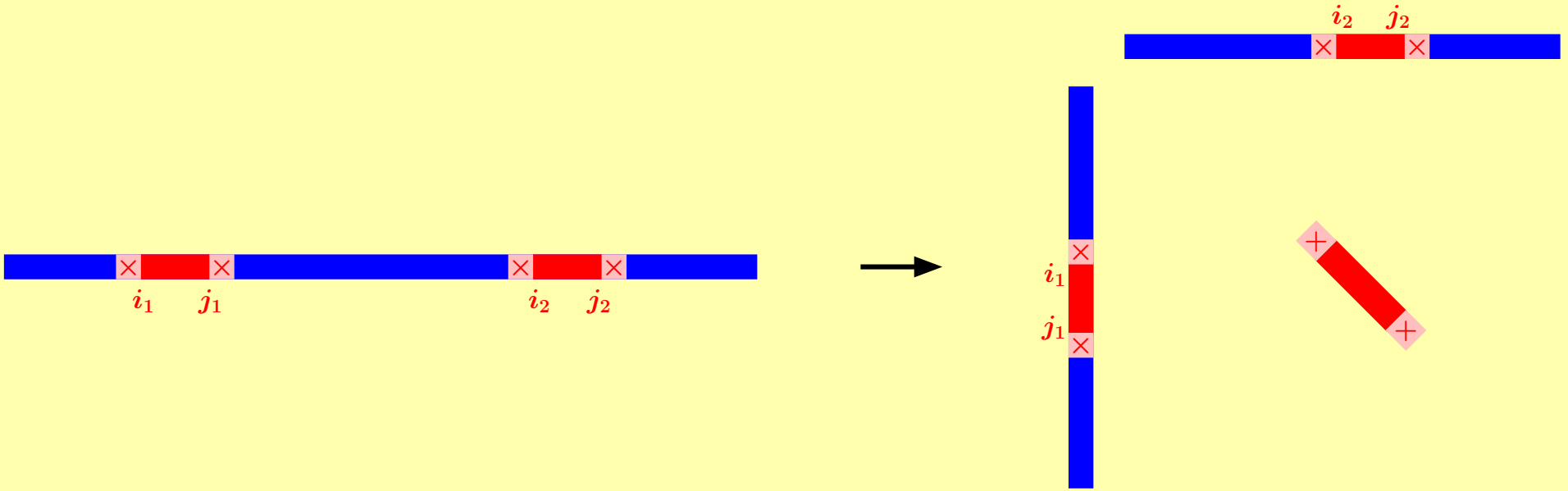
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



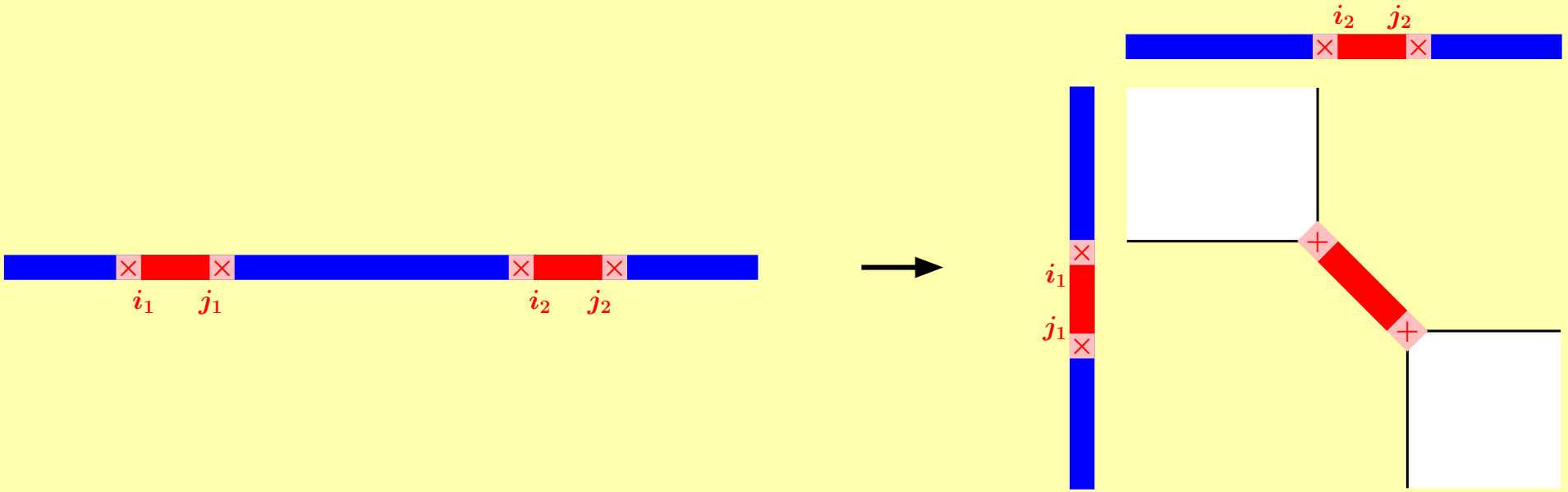
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



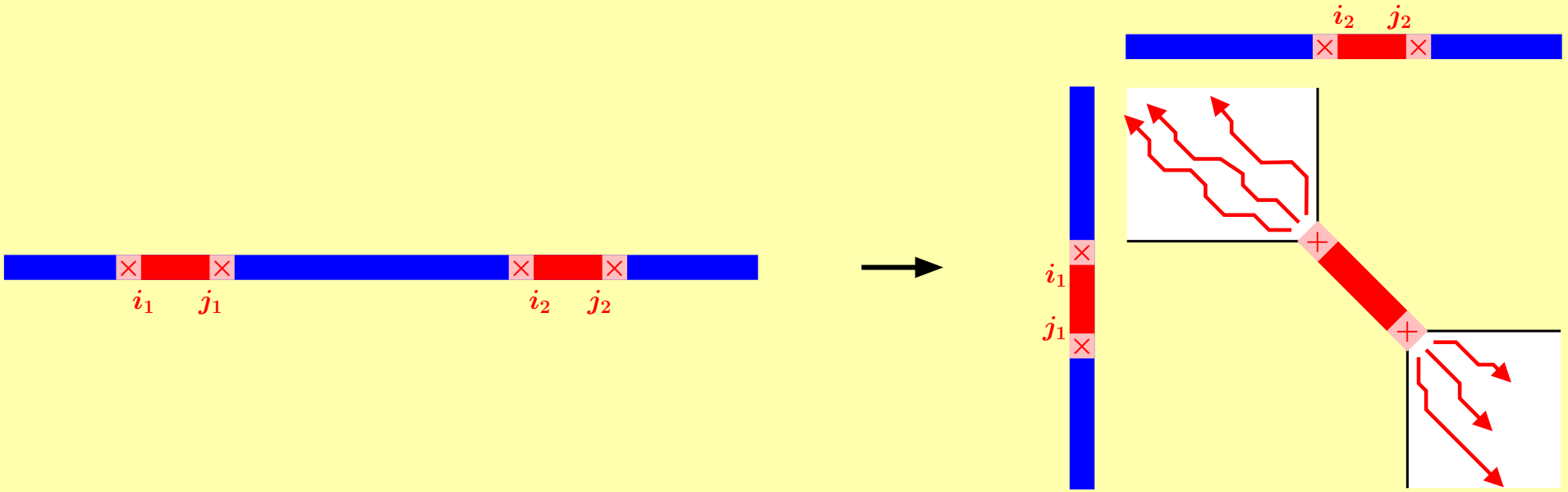
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



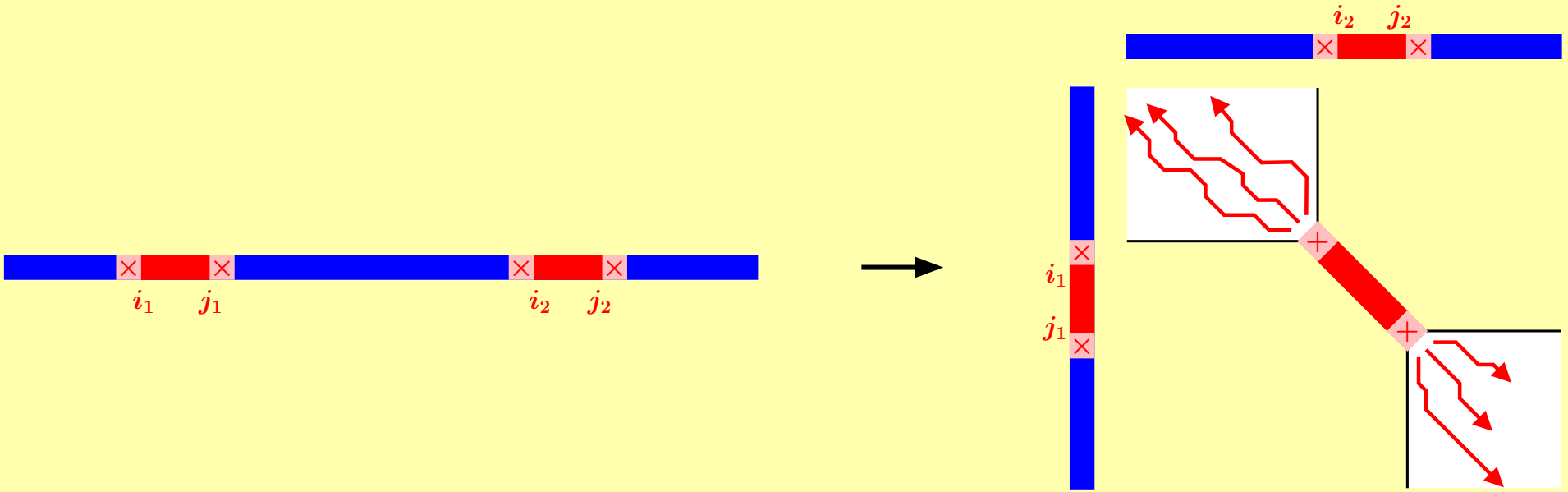
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



Analysis: $\mathcal{O}(n + \zeta k^3)$ time with $E(\zeta) = O\left(n^2/4^s\right)$, s minimal seed length.

Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

The *REP*uter suite of repeat finding programs

(joint work with S. Kurtz, E. Ohlebusch, R. Giegerich, C. Schleiermacher, J. Choudhuri)

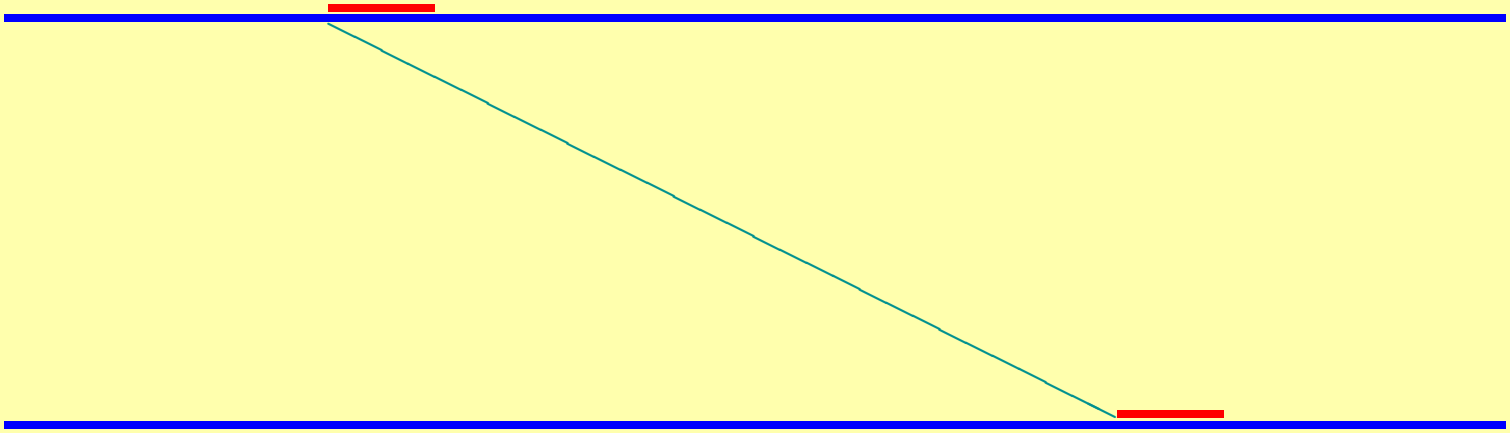
`www.genomes.de`

- *REP*find: implements several of the described algorithms.
- *REP*select: selects interesting repeats from the output of *REP*find (user-defined second filter phase).
- *REP*vis: interactive visualization tool to display large amounts of repeat data.

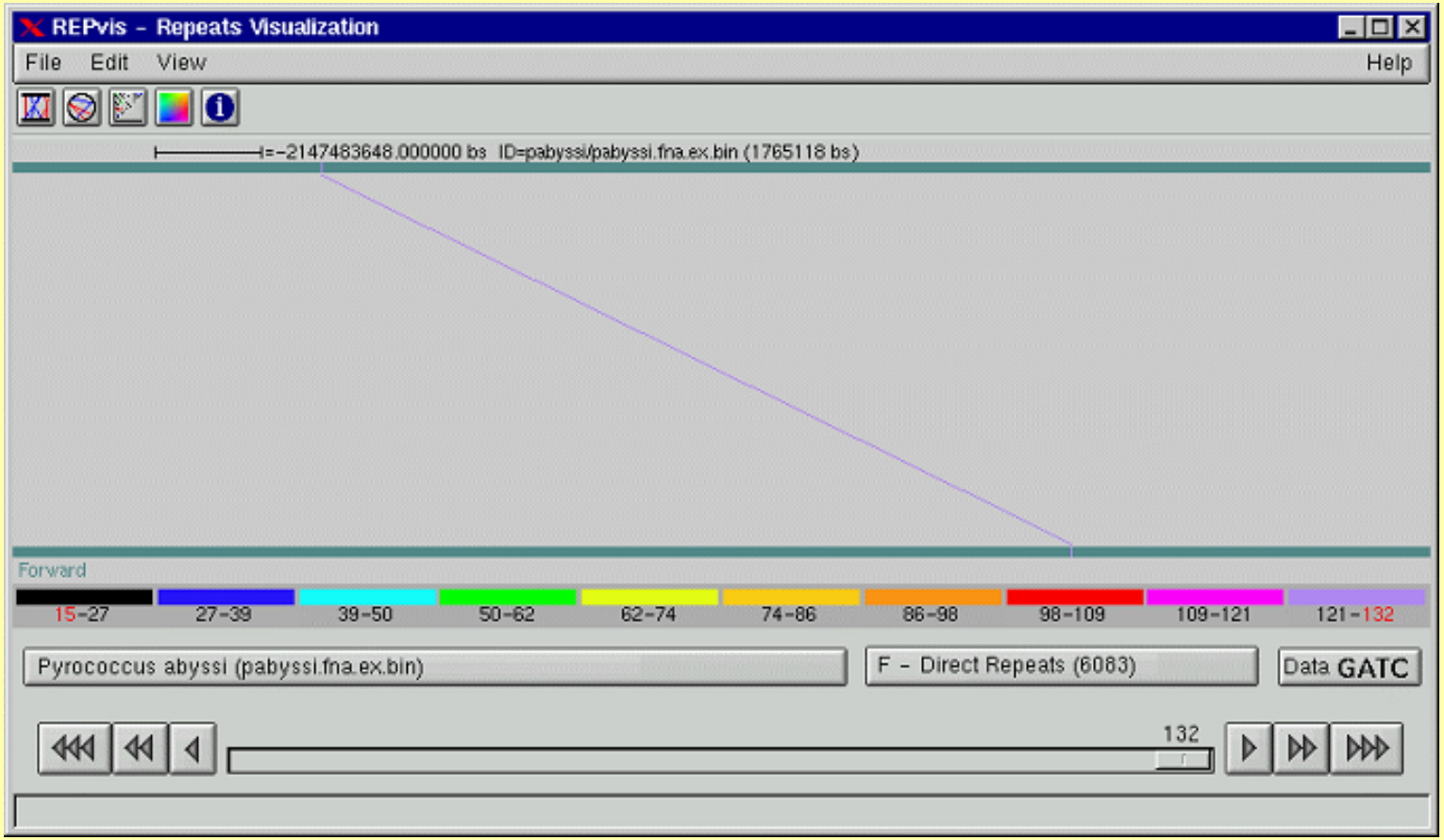
*REP*uter: An example

*REP*uter: An example

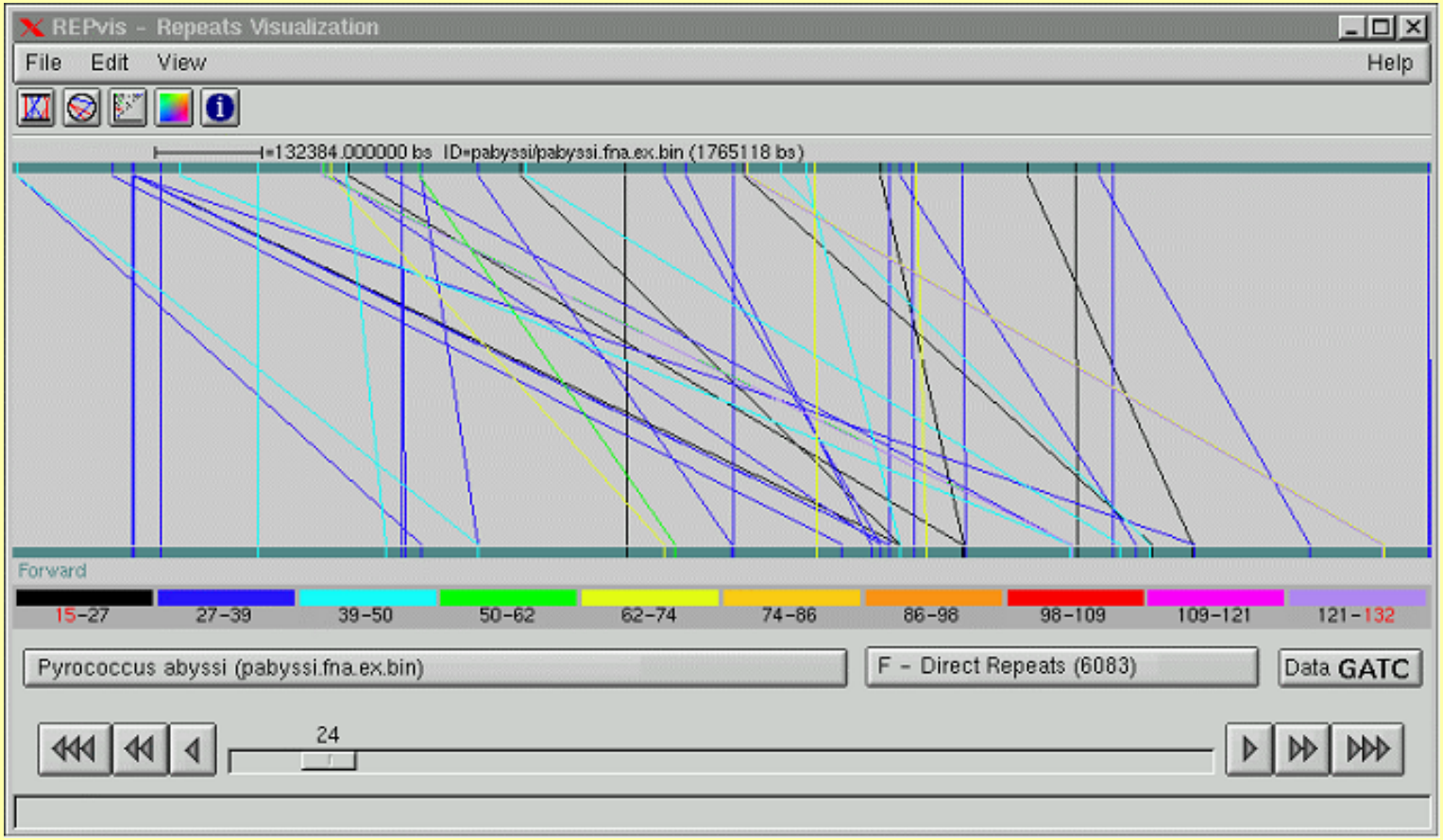
*REP*uter: An example



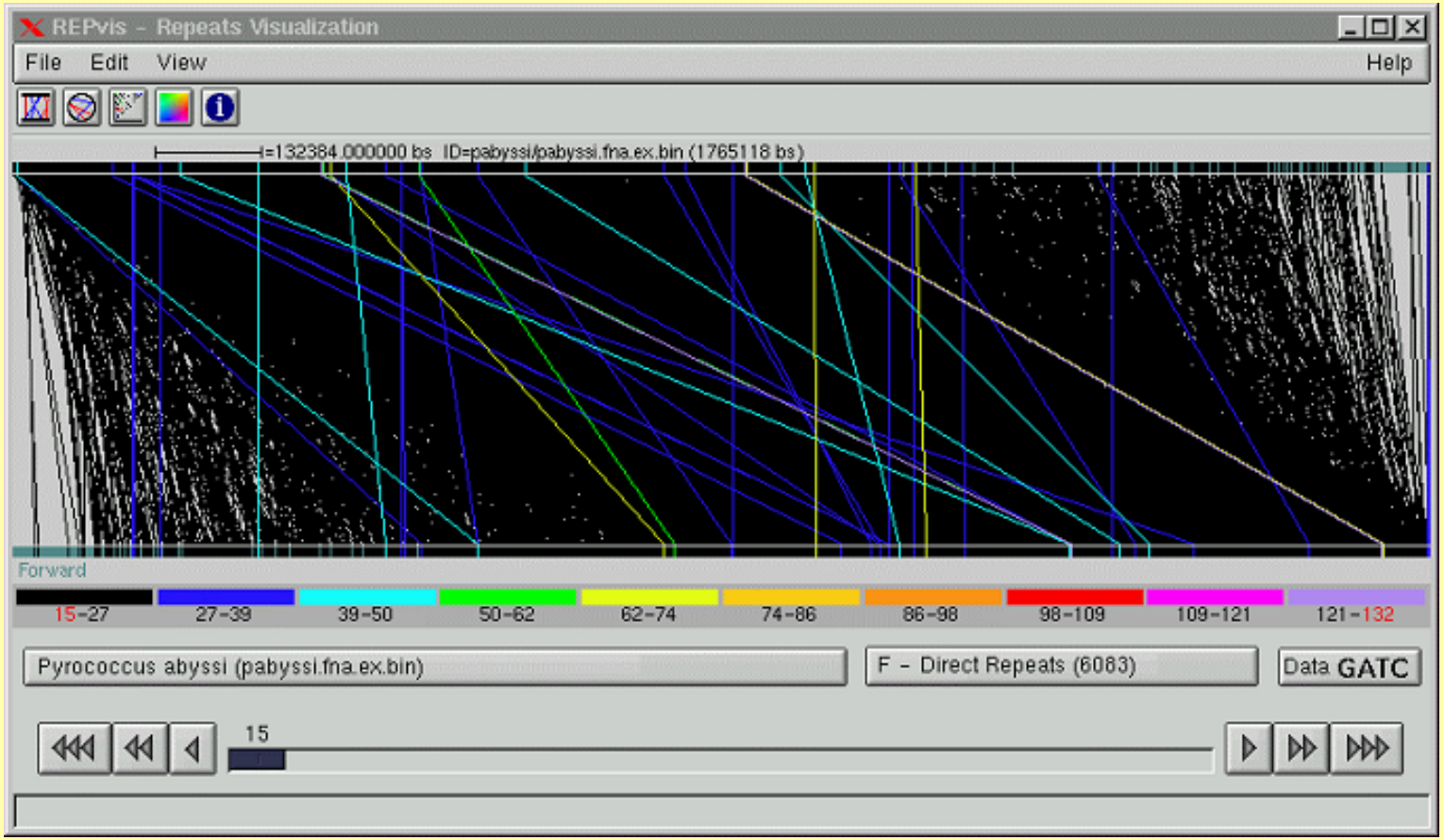
*REP*uter: An example



*REP*uter: An example

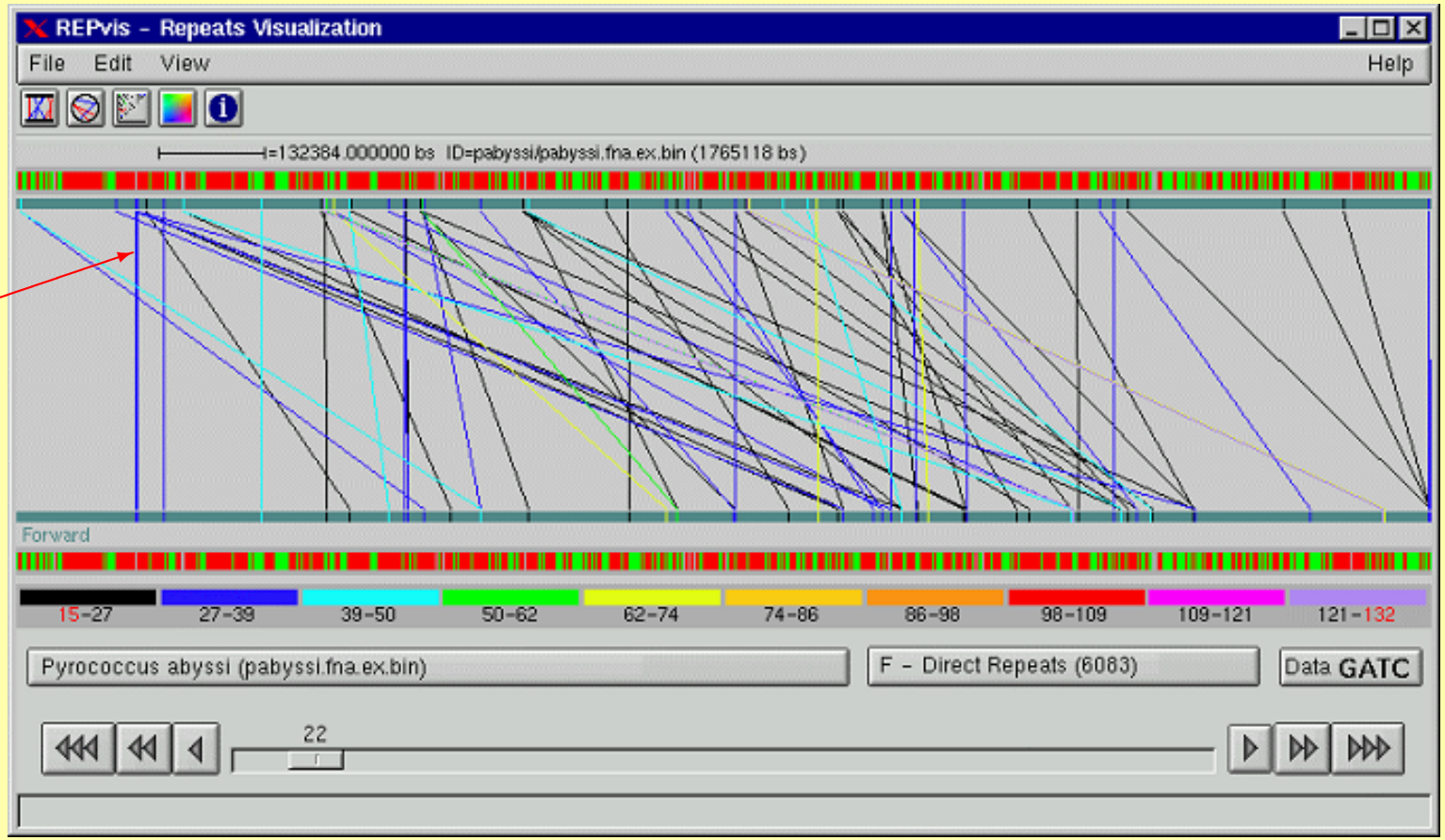


*REP*uter: An example

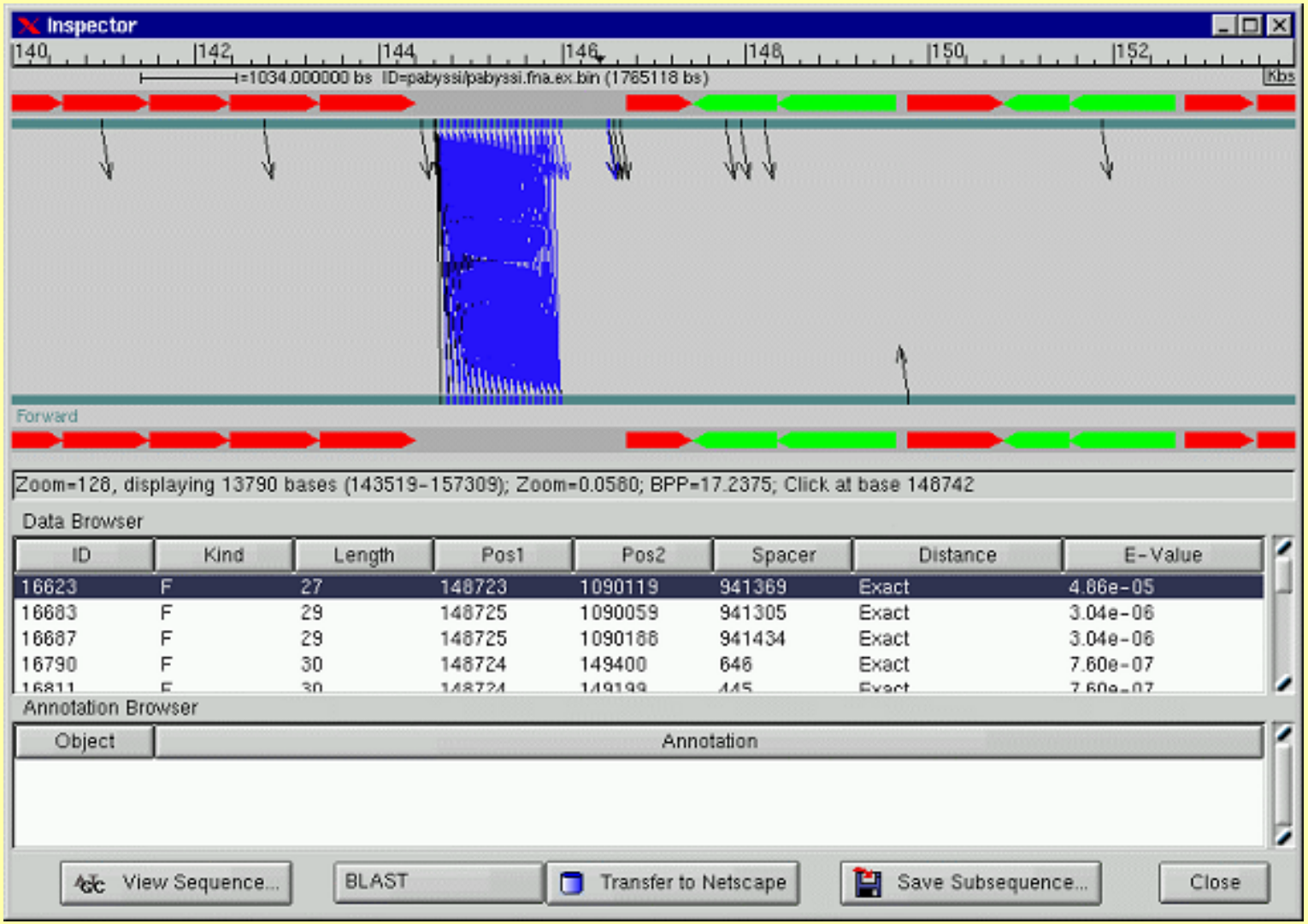


*REP*uter – Application 1: (Approximate) tandem array

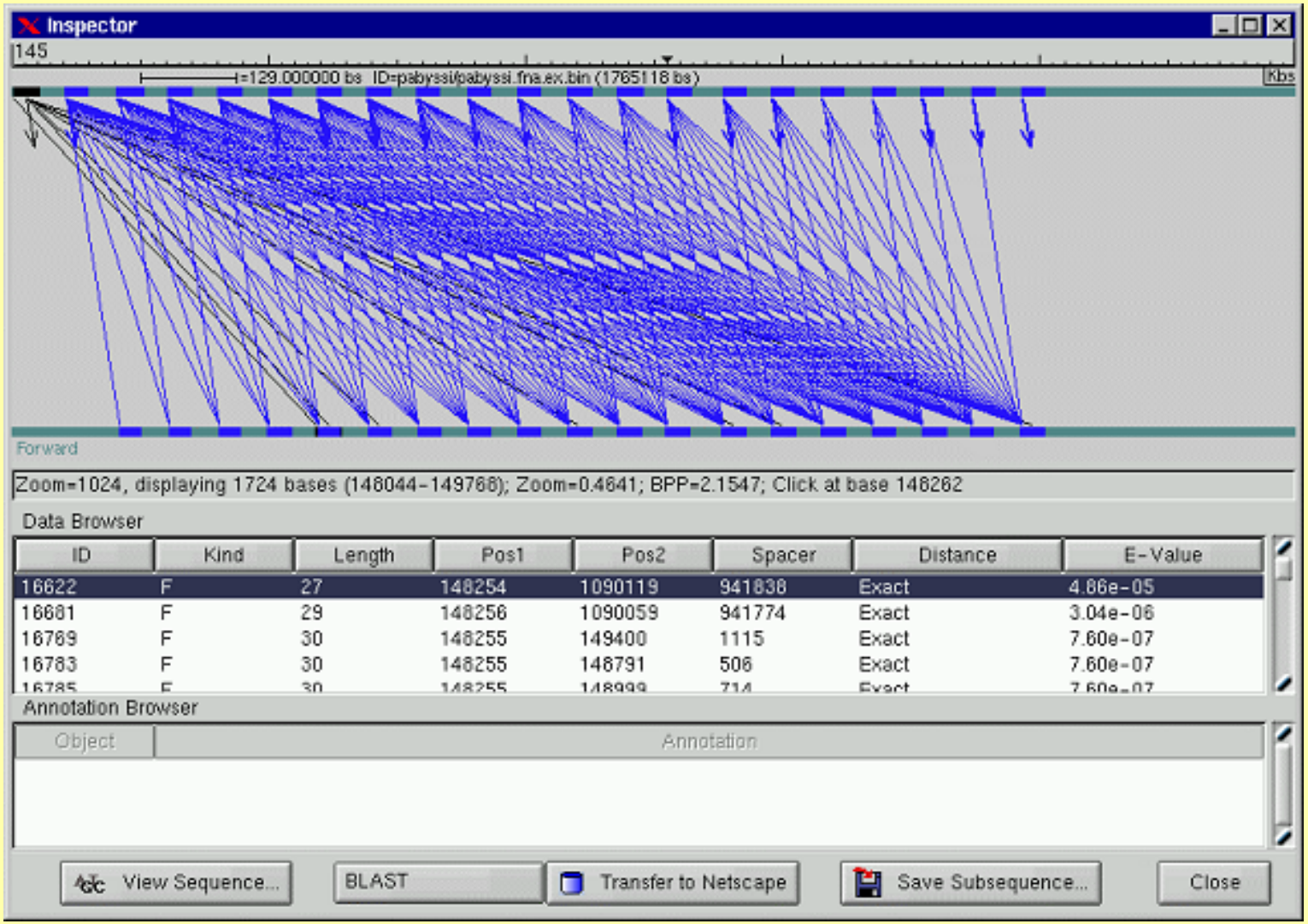
*REP*uter – Application 1: (Approximate) tandem array



*REP*uter – Application 1: (Approximate) tandem array

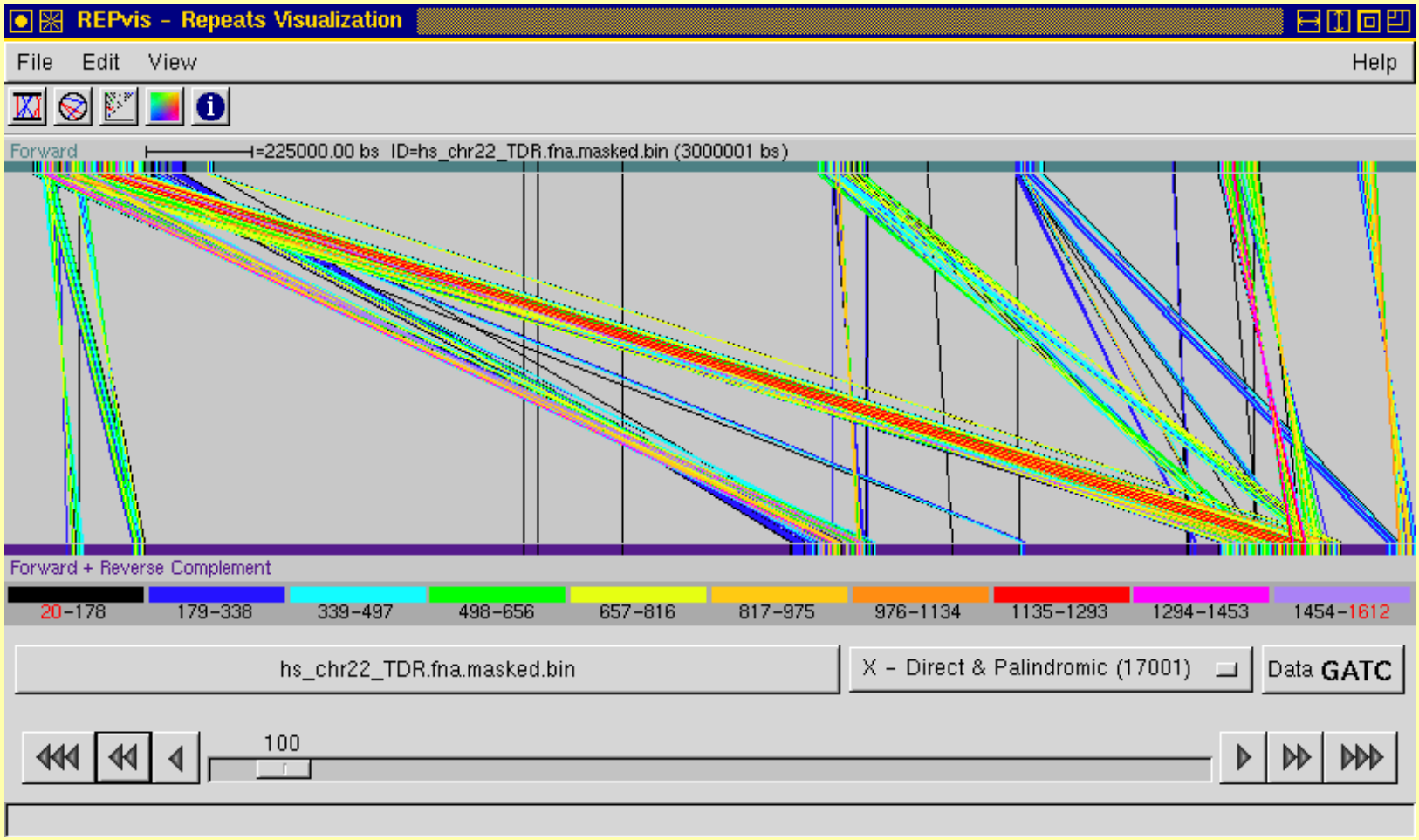


*REP*uter – Application 1: (Approximate) tandem array



*REP*uter – Application 2: Low copy repeats

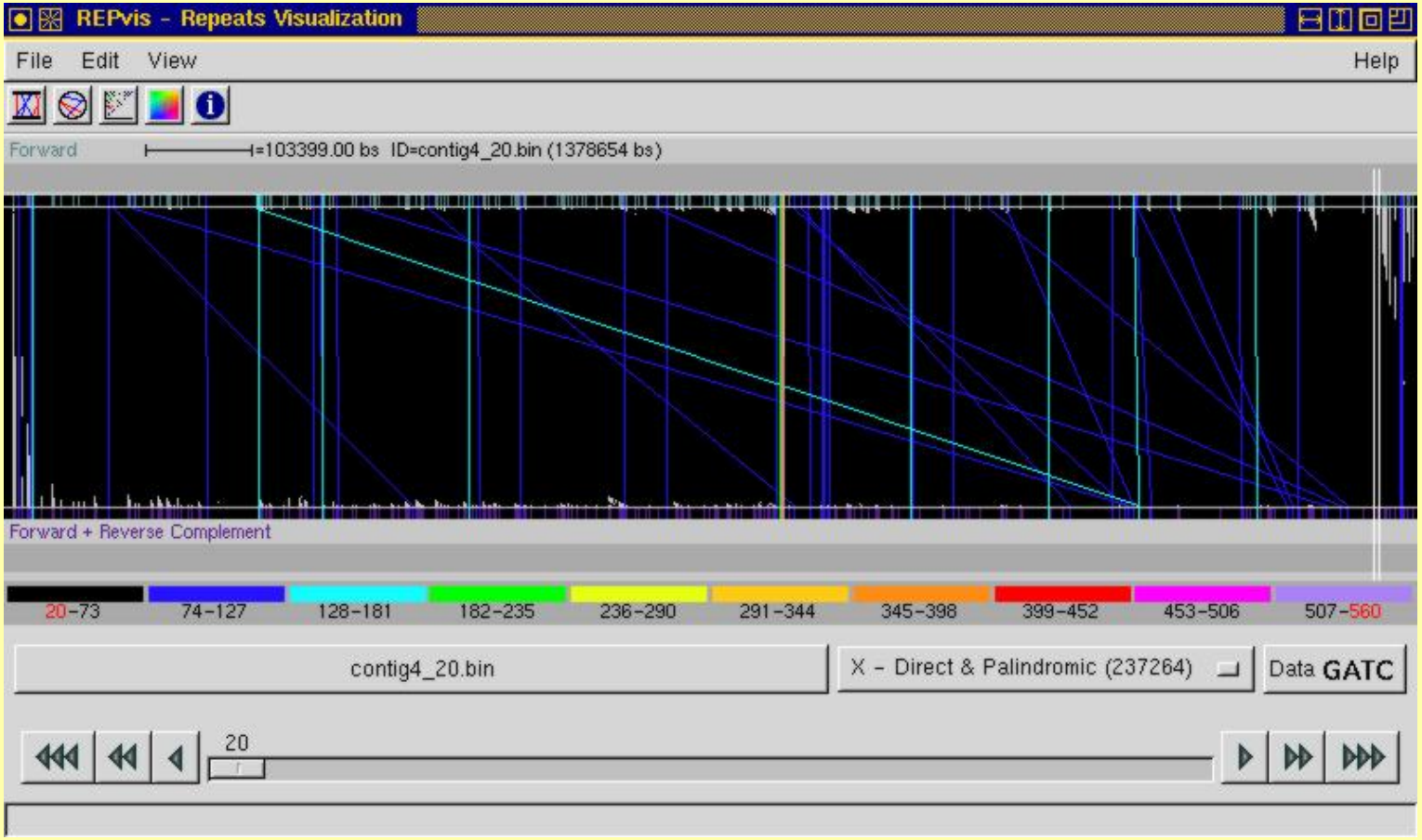
*REP*uter – Application 2: Low copy repeats



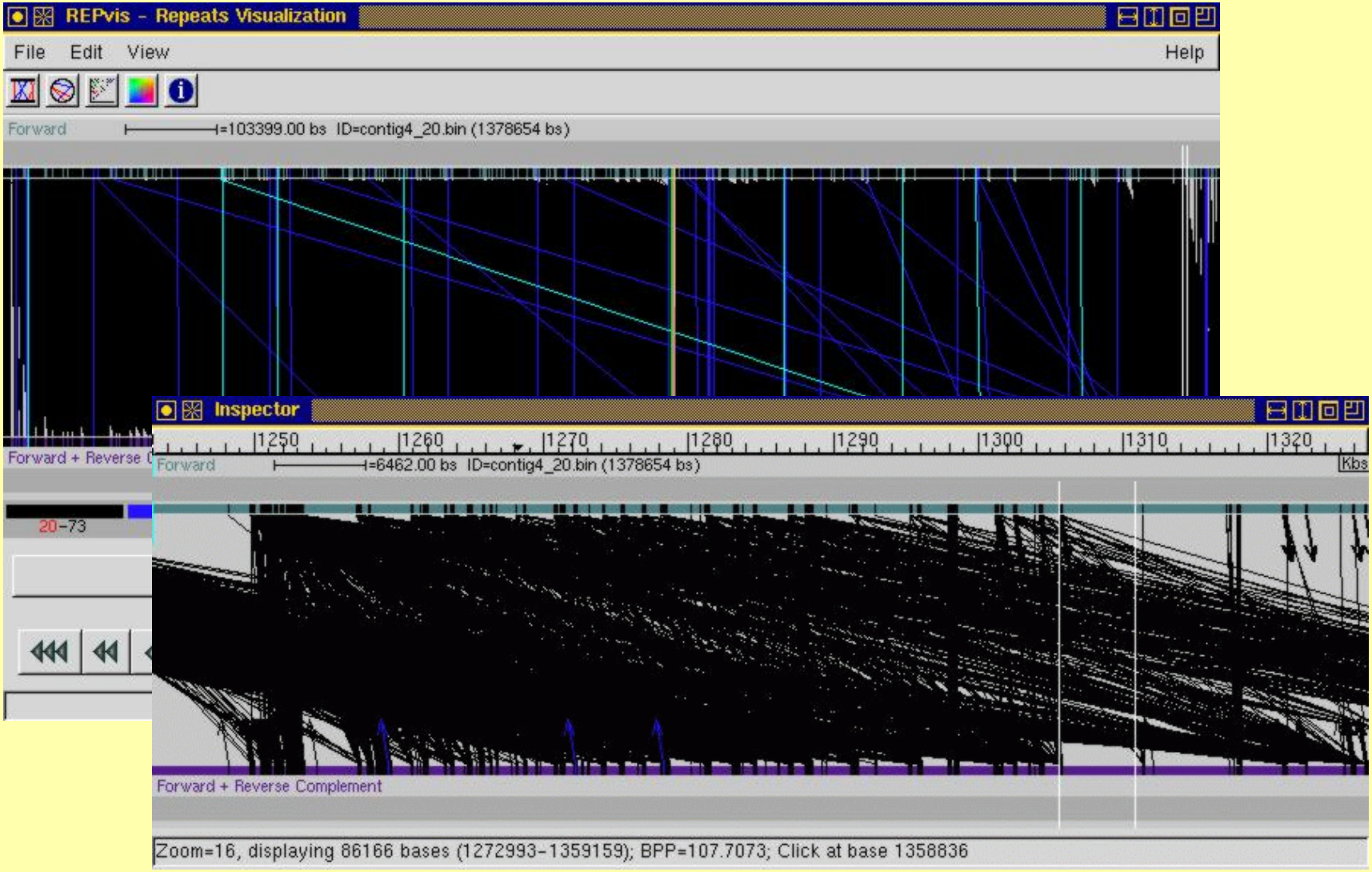
(22q11.2 region of human chromosome 22, associated to DiGeorge/Velo-cardio-facial syndrome.)

*REP*uter – Application 3: Unique sequences

*REP*uter – Application 3: Unique sequences



*REP*uter – Application 3: Unique sequences



*REP*uter: Computation times

genome	size [Mbases]	l [bases]	suffix tree [sec]	virtual tree [sec]
E. coli	4.42	150	5.4	1.7
S. cerevisiae	11.50	180	14.8	4.7
D. melanogaster	114.44	700	310.7	44.4

virtual (suffix) tree = suffix array, enhanced by functions to simulate suffix tree functionality $\longrightarrow$ GENalyzer.

Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Multiple Genome Aligner

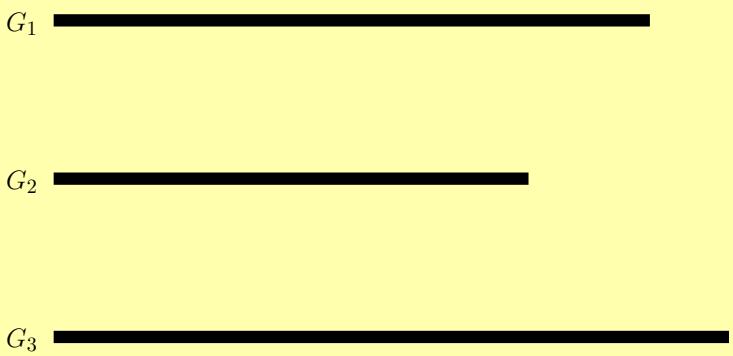
Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).

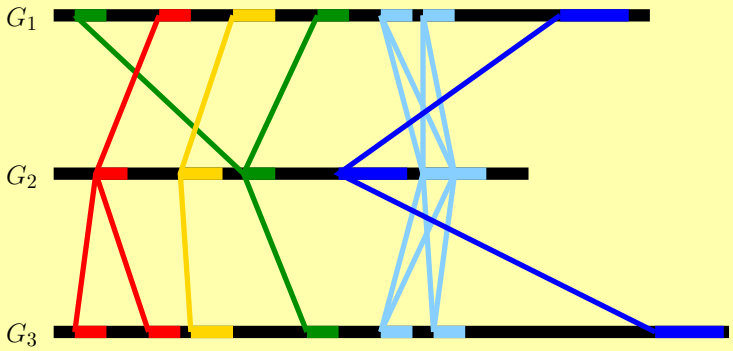


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).

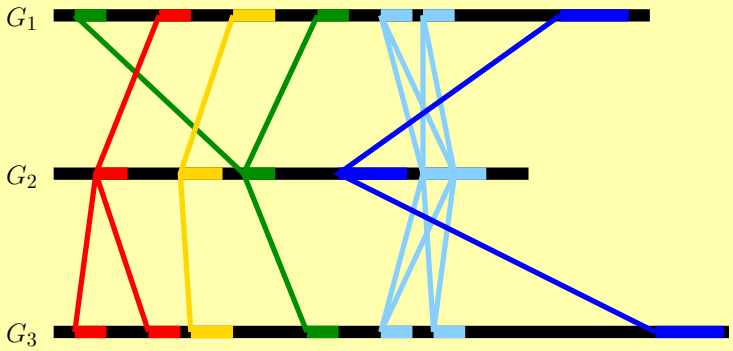


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

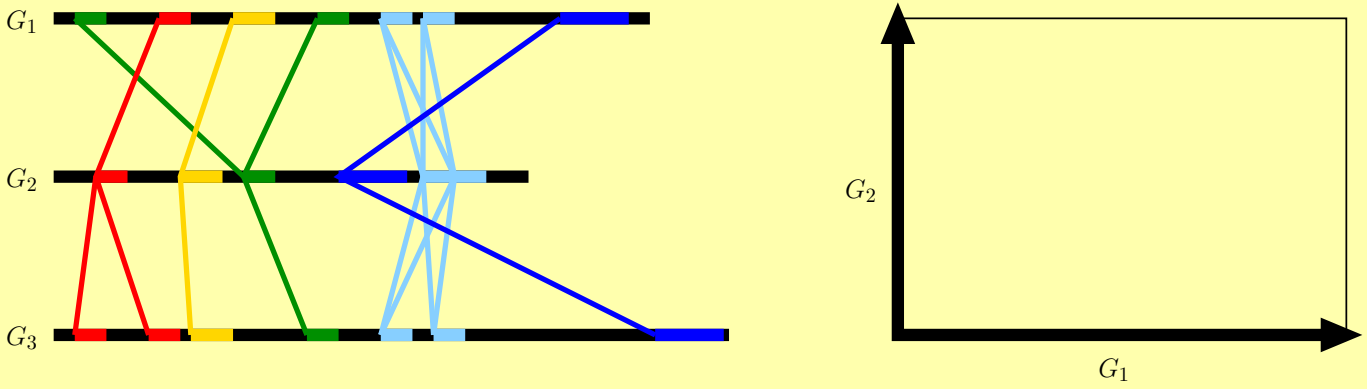


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

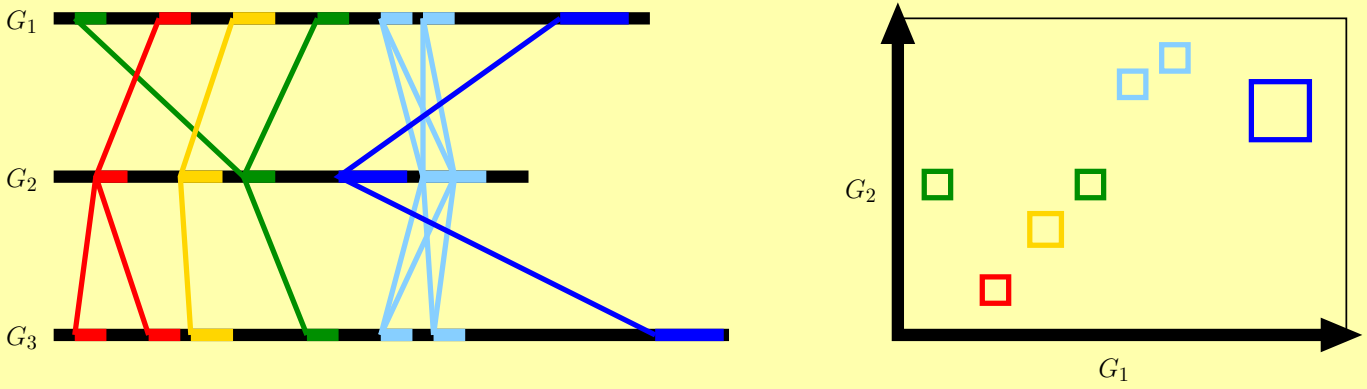


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

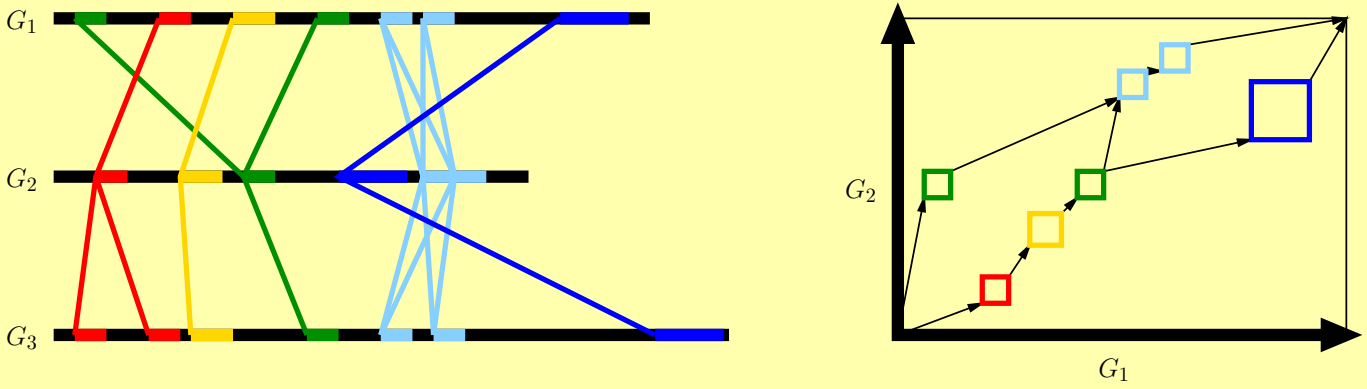


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

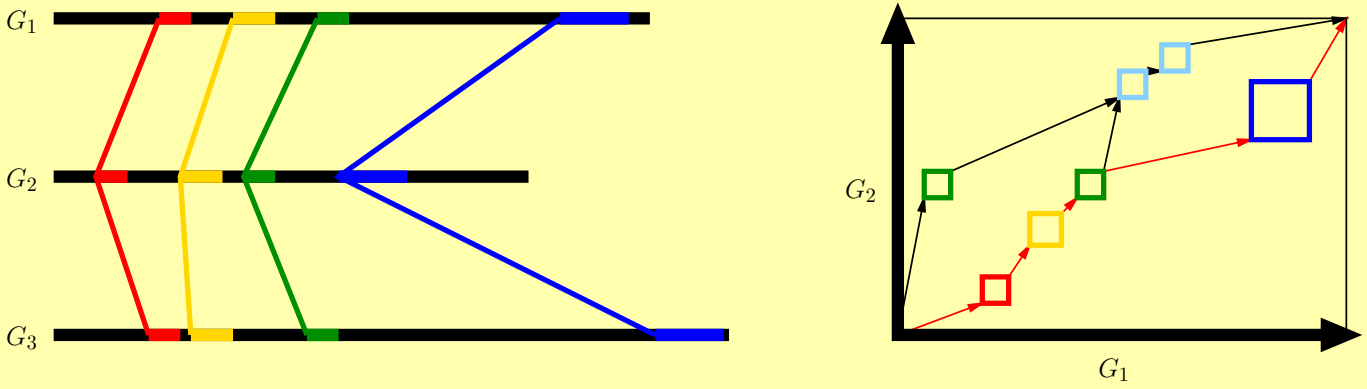


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

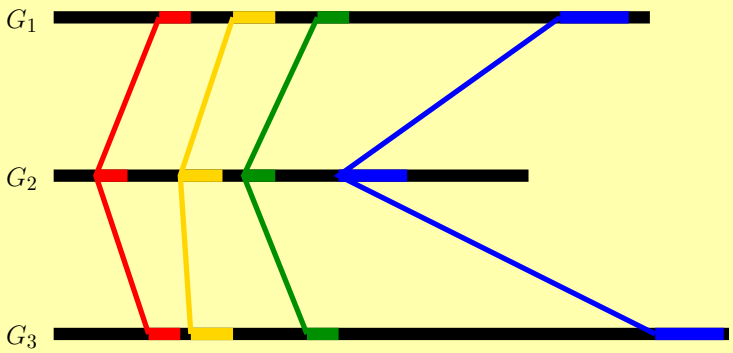


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

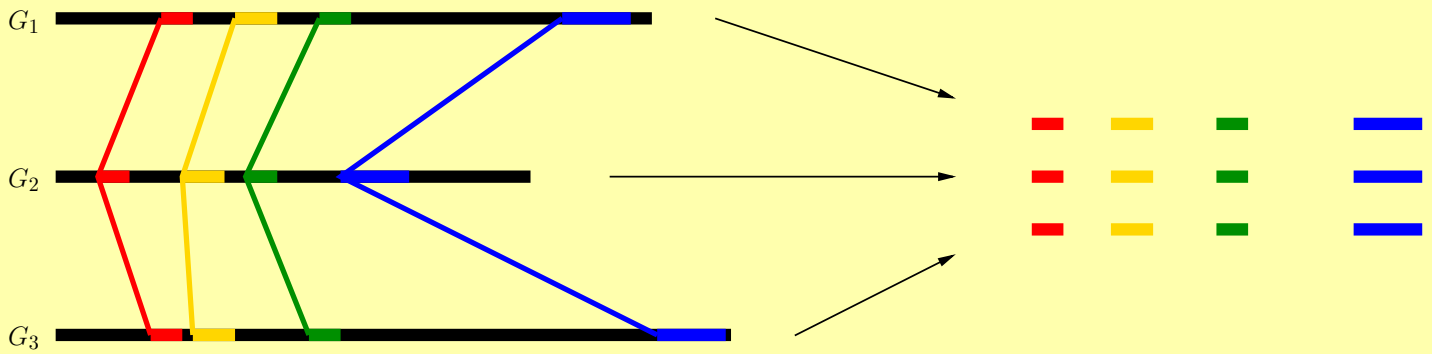


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.
3. Close the gaps recursively, and finally by a standard alignment procedure.

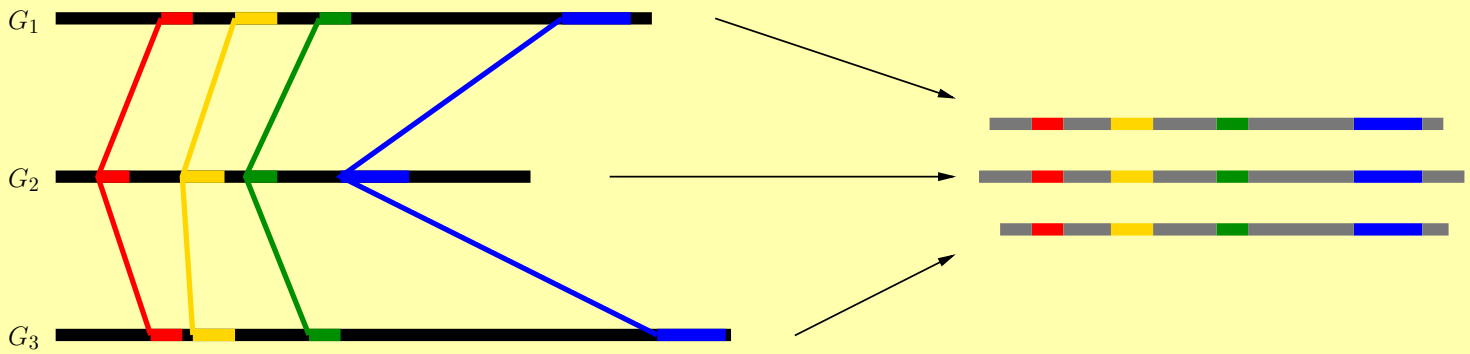


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.
3. Close the gaps recursively, and finally by a standard alignment procedure.



Overview: Finding repetitive structures in large sequences

- Introduction
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats (squares)
  - More general repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Summary: Repeats and suffix trees

Some results:

- Find all maximal repeats in $\mathcal{O}(n + |\text{output}|)$ time.
- Find all maximal palindromic repeats in $\mathcal{O}(n + |\text{output}|)$ time.
- Find all tandem repeats in $\mathcal{O}(n \log n + |\text{output}|)$ time or $\mathcal{O}(n + |\text{output}|)$ time.
- Find all maximal repeats with bounded gap in $\mathcal{O}(n \log n + |\text{output}|)$ time.
- Find all maximal repeats with lower-bounded gap in $\mathcal{O}(n + |\text{output}|)$ time.
- Find all degenerate repeats with $\leq k$ errors in $\mathcal{O}(n + \zeta k^3)$ time $(E(\zeta) = \mathcal{O}\left(n^2/4^s\right))$.

Conclusion

The analysis of biological sequence data produces several interesting computational questions.

Various CS disciplines are involved:

- Algorithms and data structures
- Algorithm engineering
- Software engineering
- Visualization

Not only does Biology profit from Computer Science, but also vice versa!

Acknowledgments

Suffix trees

- Stefan Kurtz (Hamburg)
- Robert Giegerich (Bielefeld)

Repeats

- Dan Gusfield (Davis)
- Christian N. S. Pedersen, Gerth S. Brodal (Århus)
- Rune B. Lyngsø (Oxford)
- Enno Ohlebusch (Ulm)
- Chris Schleiermacher (Artemis Pharmaceuticals, Köln)
- Jomuna Choudhuri (Bielefeld)

Black slide