Repeat Analysis on a Genomic Scale

Jens Stoye

Technische Fakultät, Universität Bielefeld, Germany

Overview

- Introduction
  - String pattern matching
  - Suffix trees (reminder)
  - Repeats
- Repeat finding with suffix trees
  - Maximal repeats
  - Tandem repeats
  - Maximal repeats with bounded gap
  - Degenerate maximal repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Pattern matching in biological sequence analysis

Finding *known* patterns: (exact/approximate)

- Search for homologous proteins
  - assumption: similar sequence ⟶ similar structure ⟶ similar function
- Mapping of *expressed sequence tags* (ESTs) onto genomic DNA
  - find location of exons/introns and alternative splicing
- Search for *repeats*, *low complexity regions* for further exclusion from analysis

Finding *structural* patterns: (exact/approximate)

- *Ab initio* gene prediction (start/stop codons, exons/introns)
- Search for over-/underrepresented subsequences, for example
  - unknown promoter binding sites
  - repeats, tandem repeats
  - possible DNA methylation sites
- Calculation of RNA secondary structure
- Sequence assembly

Requirements for computational tools

- Efficiency: linear in space and time
- Flexibility: applicable to as many problems as possible
- Statistical assessment of significance of results
- Visualization

For routine tasks, even linear time is intolerable ⟶ index

Many indices for massive sequence data use the property that the text is partitioned into words (e.g. natural language, syntactic tags).

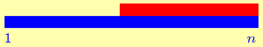
Genomic data is not divided into obvious "words".

We need an index that allows access to any substring of the text.

⟶ Suffix Tree

Suffix Tree: Definition

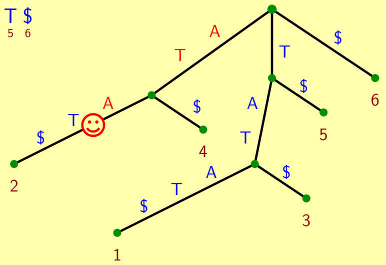
- A suffix of a string S of length n is a substring of S that ends at position n.



- 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.

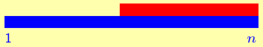
$S =$ T A T A T \$ (1 2 3 4 5 6)

$P =$ A T A



Suffix Tree: Definition

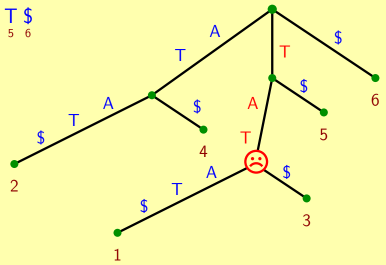
- A suffix of a string S of length n is a substring of S that ends at position n.



- 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.

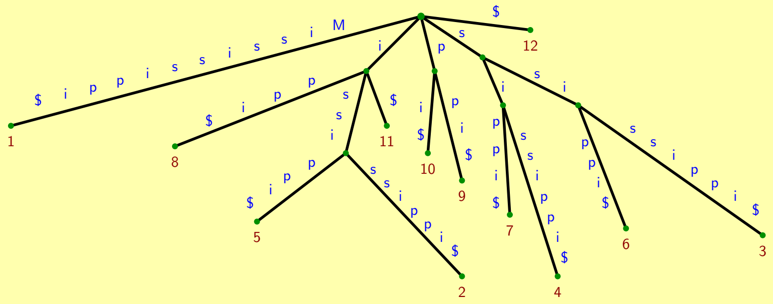
$S =$ T A T A T \$ (1 2 3 4 5 6)

$P =$ T A T T



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)



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:

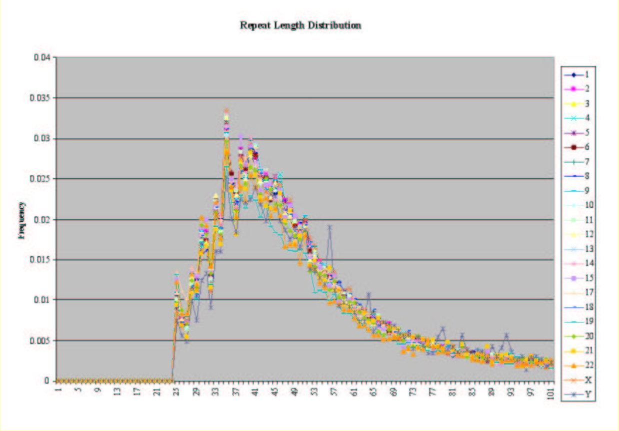
- A. Apostolico: The myriad virtues of subword trees, 1985.
- D. Gusfield: Algorithms on strings, trees, and sequences, 1997.

Repeats in biosequence analysis

- DNA of eukaryotes is highly repetitive.
  - 30% 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, satellites, ...
- 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.

⇒ Repeats are very important when studying genomic DNA.

Repeats in the human genome

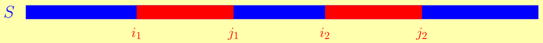


(Human Genome Sequencing Center, Baylor College of Medicine, Houston, Texas)

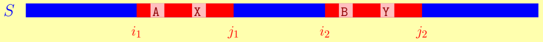
Basic definitions

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

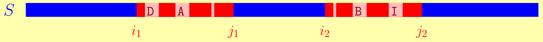
⟶ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$



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



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



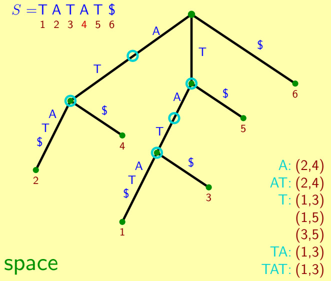
Finding exact repeats

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

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

Idea:

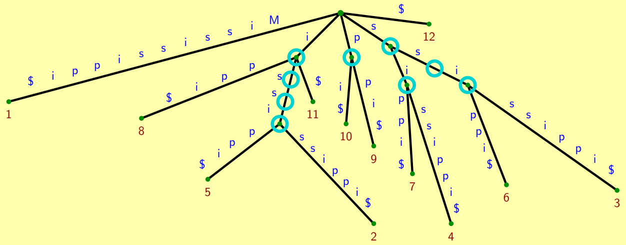
- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.



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

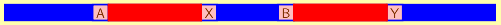
A larger example

$S =$ M i s s i s s i p p i $
1 2 3 4 5 6 7 8 9 10 11 12



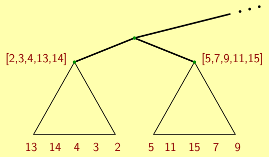
i: (8,5)	is: (5,2)	p: (10,9)	s: (7,4)	si: (7,4)
(8,2)			(7,6)	
(8,11)	iss: (5,2)		(7,3)	ss: (6,3)
(5,2)			(4,6)	
(5,11)	issi: (5,2)		(4,3)	ssi: (6,3)
(2,11)			(6,3)	

Finding *maximal* exact repeats

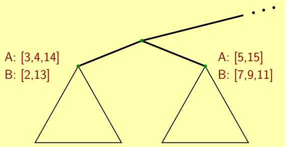


Idea:

- 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)



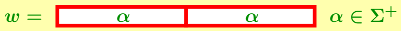
- For left-maximality (A $\neq$ B)
  - keep lists for the different left-characters
  - report only pairs from different lists



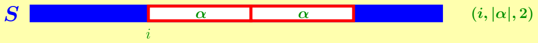
Analysis: $O(n+z)$ time with $z = |\text{output}|$, $O(n)$ space

Tandem repeats: Definitions

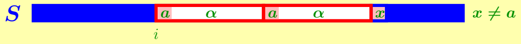
- tandem repeat (square)



- occurrence of a tandem repeat



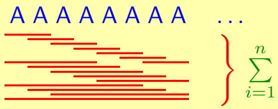
- (right-) branching occurrence of a tandem repeat



- 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

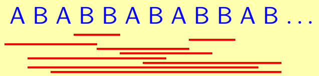
Tandem repeats: Numbers

- number of occurrences of tandem repeats: $O(n^2)$



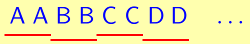
$$\sum_{i=1}^{n}(n-2i+1) = \frac{n^2}{4}$$

- number of occurrences of primitive tandem repeats: $O(n \log n)$



(Fibonacci string)

- number of different strings that occur as tandem repeats: $O(n)$



Finding tandem repeats: Overview

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

- Main/Lorentz, 1979/1984
- Landau/Schmidt, 1993

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

- Crochemore, 1981
- Apostolico/Preparata, 1983

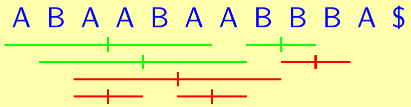
C. Find all occurrences of primitive tandem *arrays* in a string.

Here:
Simple and flexible detection of all of these in optimal time using a suffix tree.
(TCS 2002, joint work with Dan Gusfield)

Basic observation

Lemma:
Any non-branching occurrence $(i, l, 2)$ of a tandem repeat is the left-rotation of another tandem repeat $(i + 1, l, 2)$, starting one position to its right.

Example:



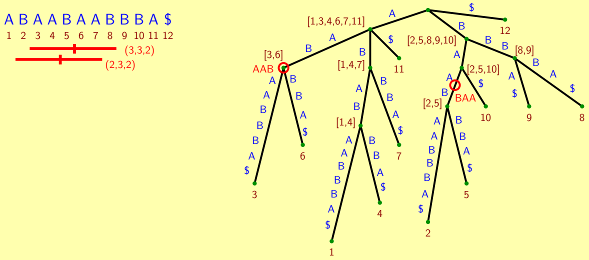
Suffix trees and tandem repeats

Lemma: (folklore)
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) \geq l$.

Example:



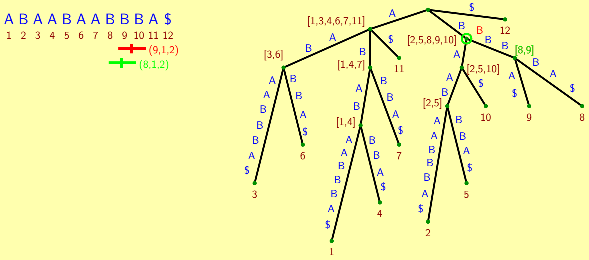
Suffix trees and *branching* tandem repeats

Lemma:
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a branching tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) = l$, but do not appear in the same leaf-list of any node with depth greater than l.

Example:



Basic algorithm

Idea:
For each node v of $T(S)$, test if $\alpha\alpha = L(v)L(v)$ is a branching tandem repeat.

Algorithm
All nodes of $T(S)$ begin unmarked.
Step 1 is repeated until all nodes are marked.

1. Select an unmarked internal node v.
   Mark v and execute steps 2a and 2b for node v.

2a. Collect the leaf-list $LL(v)$.

2b. For each leaf i in $LL(v)$, test whether the leaf $j = i + D(v)$ is in $LL(v)$.
If so, test whether $S[i] \neq S[i + 2D(v)]$. There is a branching tandem repeat of length $2D(v)$ starting at position i if and only if both tests return true.

Analysis: $O(n^2)$ time, $O(n)$ space

Testing in constant time

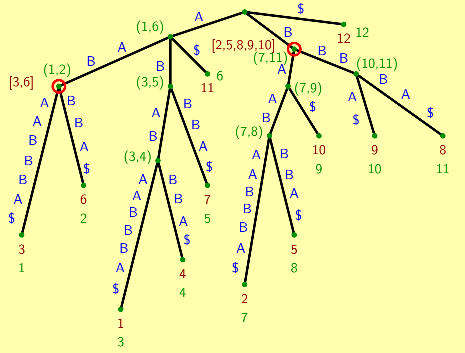
Depth-first numbering and look-up table:

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

B: $8 \xrightarrow{D=1} 9? \xrightarrow{table} 10 \in (7,11)$
but: $(8,9)$ not branching

$9 \xrightarrow{D=1} 10? \xrightarrow{table} 9 \in (7,11)$
and: $(9,10)$ is branching

AAB: $3 \xrightarrow{D=3} 6? \xrightarrow{table} 2 \in (1,2)$
and: $(3,6)$ is branching

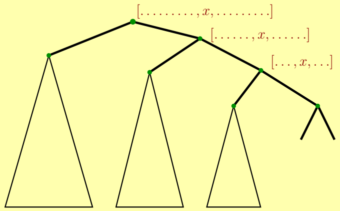


Speedup of the basic algorithm

Definitions

- For each node v, v' denotes the child of v with the largest leaf-list.
- $LL'(v)$ denotes $LL(v) - LL(v')$.

The "Smaller Half" Trick
It is well known that $\sum_v |LL'(v)| \leq n \log_2 n$.



$[\ldots\ldots\ldots, x, \ldots\ldots\ldots]$
$[\ldots\ldots, x, \ldots\ldots]$
$[\ldots, x, \ldots]$

Any value x can be
in at most $\log_2 n$ leaf-lists LL'.

Optimized basic algorithm

Algorithm
All nodes of $T(S)$ begin unmarked.
Step 1 is repeated until all nodes are marked.

1. Select an unmarked internal node v.
   Mark v and execute steps 2a, 2b and 2c for node v.

2a. Collect the list $LL'(v)$ for v.

2b. For each leaf i in $LL'(v)$, test whether leaf $j = i + D(v)$ is in $LL(v)$, the leaf-list of v. If so, test whether $S[i] \neq S[i + 2D(v)]$. There is a branching tandem repeat of length $2D(v)$ starting at position i if and only if both tests return true.

2c. Do the same test for each leaf j in $LL'(v)$, and $i = j - D(v)$.

Analysis: $O(n \log n)$ time, $O(n)$ space

Extension 1: Primitive tandem repeats

Remember:

- 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

Lemma

A string ua is primitive if and only if its left-rotation au is primitive.

Extension of the Algorithm

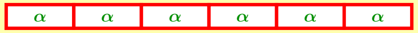
Remove marks of the non-primitive branching tandem repeats.
Do the rotation procedure only for the primitive branching tandem repeats.

Analysis: $O(n \log n)$ time, no additional space

Extension 2: Primitive tandem arrays

Definition

A primitive tandem array is a string $w = \alpha^k$ with α primitive, $k \geq 2$.



Extended Algorithm

For an observed branching occurrence of a primitive tandem repeat $(i, l, 2)$ at node v, successively test for $k = 1, 2, \ldots$ whether leaf $j = i - kl$ is also in the subtree below v.

Analysis

Right-maximal occurrences of primitive tandem arrays: $O(n \log n)$ time.
All occurrences of primitive tandem arrays: $O(n \log n + z)$ time.

Can be extended further to find only the *maximal* primitive tandem arrays in $O(n \log n)$ time.

Tandem repeats in linear time!

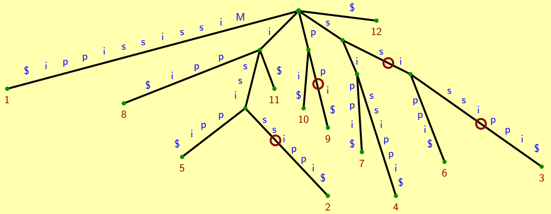
Fraenkel & Simpson, 1998:

- The vocabulary of all tandem repeats in S has only $O(n)$ elements.

Idea:

- Mark in $T(S)$ all end points of tandem repeats.

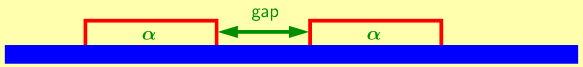
$S =$ M i s s i s s i p p i $
1 2 3 4 5 6 7 8 9 10 11 12



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

Repeats with bounded gap

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



Idea:

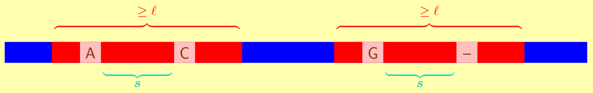
- Traverse the suffix tree bottom-up.
- At each vertex v collect the leaf-list $LL'(v)$.
- Output only pairs that have the required distance.

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

Finding *degenerate* repeats

Often, repats in genomic DNA are degenerate, i.e. at some positions more than one base is possible.

Idea: Filter method (seed and extend)



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

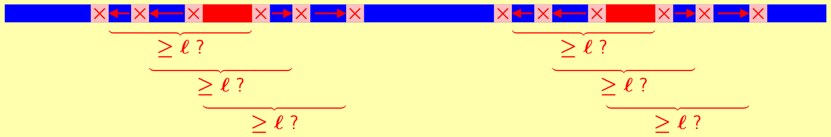
Algorithm:

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

Extension for maximal k-mismatch repeats

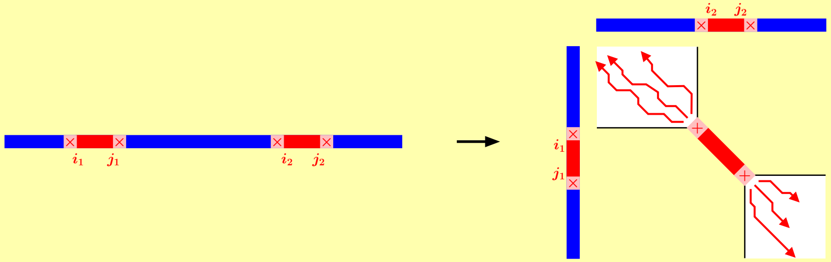
Simple extension and length test:



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

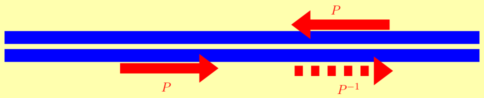
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:

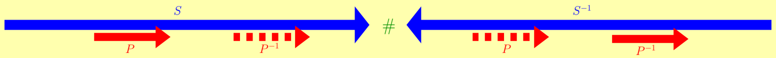


Analysis: $O(n + \zeta k^3)$ time with $E(\zeta) = O\left(n^2/4^s\right)$, s minimal seed length.

Variation: Palindromic repeats



- One repeat instance must be reverse Watson/Crick complement P^{-1}.
- Essentially same problem as computing direct repeats.
- Instead of S use $S\#S^{-1}$ (where S^{-1} is the reverse complement of S).



- $\#$ is a unique separator symbol.
- One of the duplicates must be in S and the other in S^{-1}.
- Calculate position in S^{-1} relative to the beginning of S.

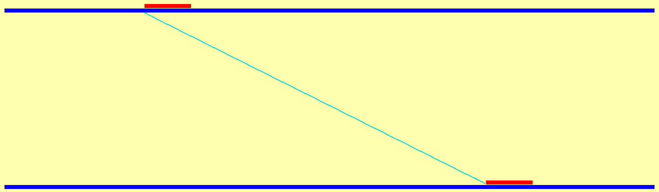
The *REP*uter suite of repeat finding programs

`www.genomes.de`

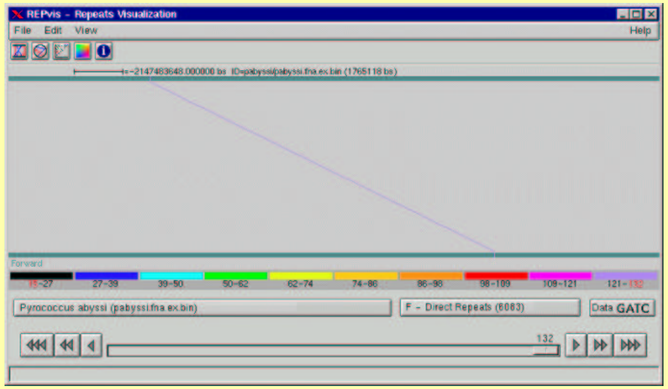
- *REP*find: implements several of the described algorithms.
- *REP*select: selects interesting repeats from the output of *REP*find (user-defined second filter phase).
- *REP*vis: interactive visualization tool to display large amounts of repeat data.

(joint work with Stefan Kurtz, Enno Ohlebusch, Robert Giegerich, Chris Schleiermacher, Jomuna Choudhuri)

*REP*uter: An example



*REP*uter: An example

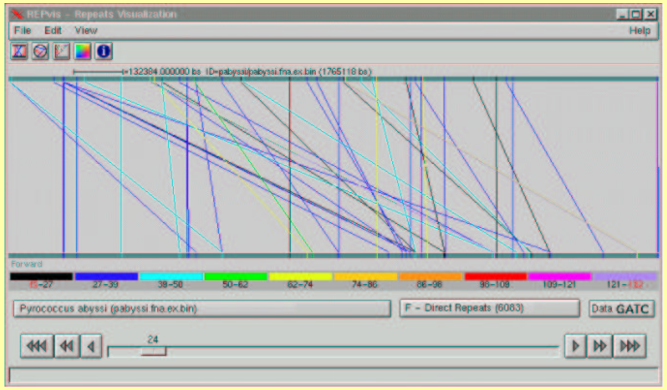


=-2147483648.000000 bs ID=pabyssi/pabyssi.fna.ex.bin (1765118 bs)

Pyrococcus abyssi (pabyssi.fna.ex.bin)

F - Direct Repeats (6083)

*REP*uter: An example

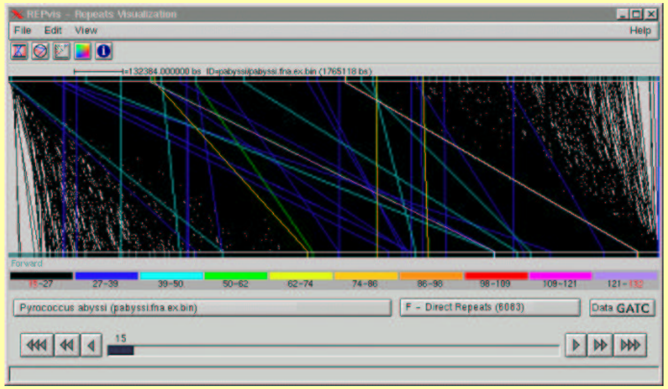


=132384.000000 bs ID=pabyssi/pabyssi.fna.ex.bin (1765118 bs)

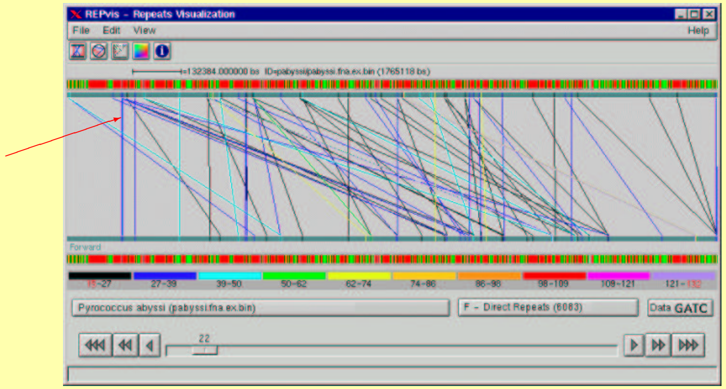
Pyrococcus abyssi (pabyssi.fna.ex.bin)

F - Direct Repeats (6083)

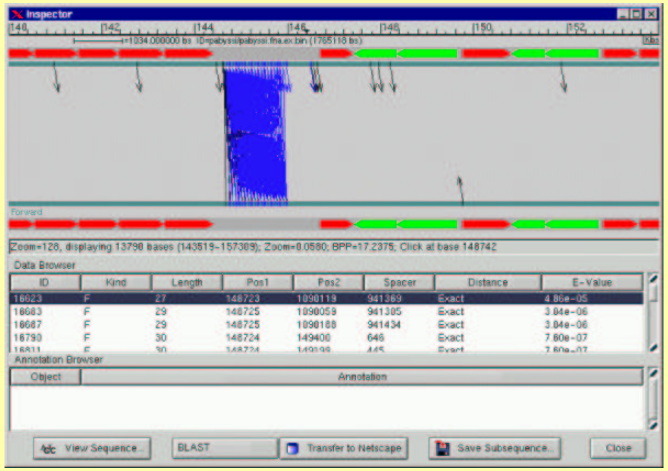
*REP*uter: An example


REPvis – Repeats Visualization
File
Edit
View
Help
=132384.000000 bs ID=pabyssi/pabyssi.fna.ex.bin (1765118 bs)
Forward
15–27
27–39
39–50
50–62
62–74
74–86
86–98
98–109
109–121
121–132
Pyrococcus abyssi (pabyssi.fna.ex.bin)
F – Direct Repeats (6083)
Data GATC
15

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


REPvis – Repeats Visualization
File
Edit
View
Help
=132384.000000 bs ID=pabyssi/pabyssi.fna.ex.bin (1765118 bs)
Forward
15–27
27–39
39–50
50–62
62–74
74–86
86–98
98–109
109–121
121–132
Pyrococcus abyssi (pabyssi.fna.ex.bin)
F – Direct Repeats (6083)
Data GATC
22

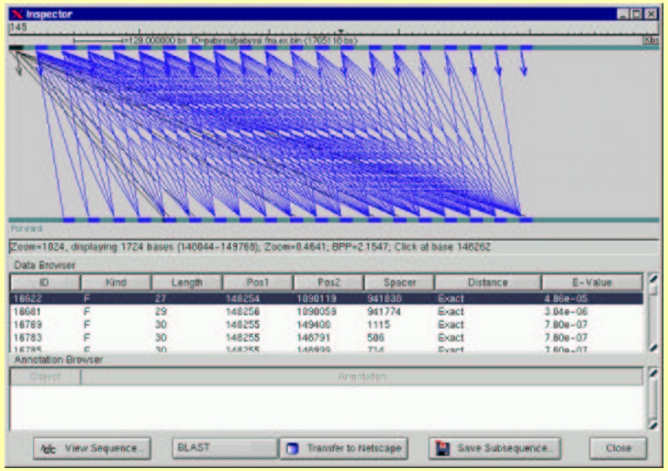
*REP*uter – Application 1: (Approximate) tandem array



Zoom=128, displaying 13790 bases (143519–157309); Zoom=0.0580; BPP=17.2375; Click at base 148742

ID	Kind	Length	Pos1	Pos2	Spacer	Distance	E-Value
16623	F	27	148723	1090119	941369	Exact	4.86e-05
16683	F	29	148725	1090059	941305	Exact	3.04e-06
16687	F	29	148725	1090188	941434	Exact	3.04e-06
16790	F	30	148724	149400	646	Exact	7.60e-07
16811	F	30	148724	149199	445	Exact	7.60e-07

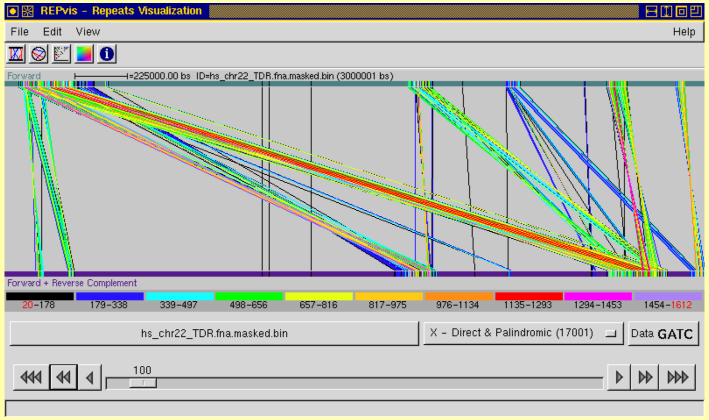
*REP*uter – Application 1: (Approximate) tandem array



Zoom=1024, displaying 1724 bases (148044–149768); Zoom=0.4641; BPP=2.1547; Click at base 148262

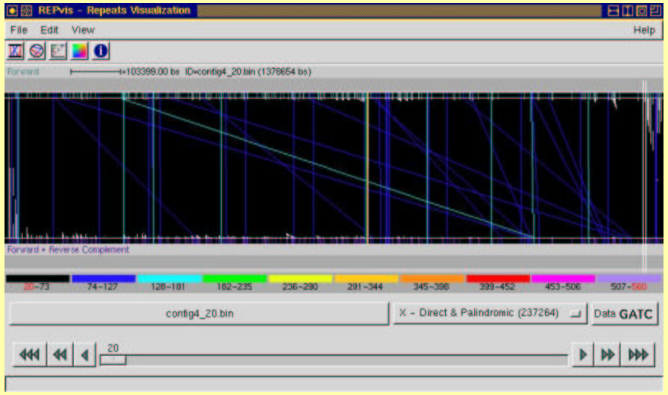
ID	Kind	Length	Pos1	Pos2	Spacer	Distance	E-Value
16622	F	27	148254	1090119	941838	Exact	4.86e-05
16681	F	29	148256	1090059	941774	Exact	3.04e-06
16769	F	30	148255	149400	1115	Exact	7.60e-07
16783	F	30	148255	148791	506	Exact	7.60e-07
16785	F	30	148255	148999	714	Exact	7.60e-07

*REP*uter – Application 2: Low copy repeats

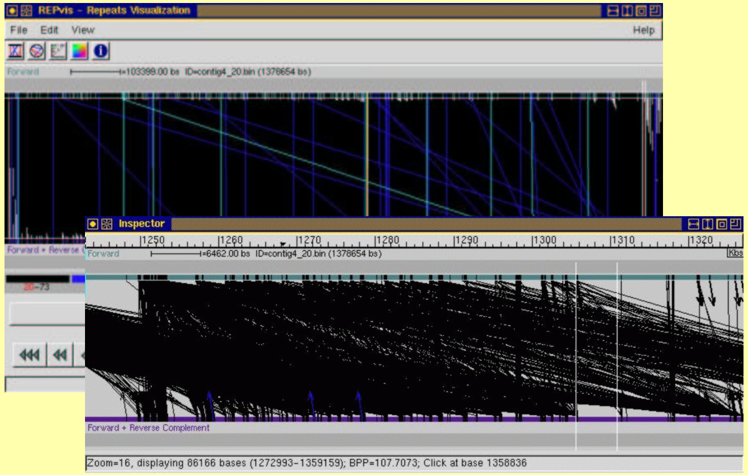

REPvis – Repeats Visualization
File
Edit
View
Help
Forward
=225000.00 bs ID=hs_chr22_TDR.fna.masked.bin (3000001 bs)
Forward + Reverse Complement
20–178
179–338
339–497
498–656
657–816
817–975
976–1134
1135–1293
1294–1453
1454–1612
hs_chr22_TDR.fna.masked.bin
X – Direct & Palindromic (17001)
Data GATC
100

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

*REP*uter – Application 3: Unique sequences


REPvis – Repeats Visualization
File
Edit
View
Help
Forward
=103399.00 bs ID=contig4_20.bin (1378654 bs)
Forward + Reverse Complement
20–73
74–127
128–181
182–235
236–290
291–344
345–398
399–452
453–506
507–560
contig4_20.bin
X – Direct & Palindromic (237264)
Data GATC
20

*REP*uter – Application 3: Unique sequences


REPvis – Repeats Visualization
File
Edit
View
Help
Forward
=103399.00 bs ID=contig4_20.bin (1378654 bs)
Forward + Reverse C
20–73
Inspector
1250
1260
1270
1280
1290
1300
1310
1320
Kbs
Forward
=6462.00 bs ID=contig4_20.bin (1378654 bs)
Forward + Reverse Complement

Zoom=16, displaying 86166 bases (1272993–1359159); BPP=107.7073; Click at base 1358836

*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

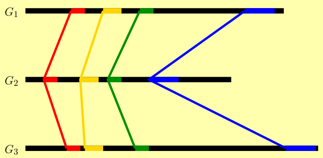
virtual (suffix) tree = suffix array, enhanced by functions to simulate suffix tree functionality → GENalyzer.

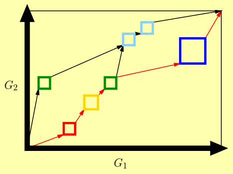
Multiple Genome Aligner

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



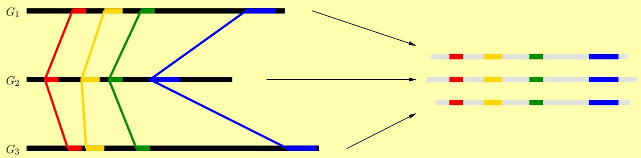


Multiple Genome Aligner

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



Summary: Repeats and suffix trees

Some results: (z is always the output size)

- Find all z maximal repeats in $O(n+z)$ time.
- Find all z maximal palindromic repeats in $O(n+z)$ time.
- Find all z tandem repeats in $O(n+z)$ time.
- Find all z maximal repeats with bounded gap in $O(n\log n+z)$ time.
- Find all z maximal repeats with lower-bounded gap in $O(n+z)$ time.
- Find all degenerate repeats with $\leq k$ errors in $O(n+\zeta k^3)$ time ($\zeta = \#$ seeds).

Conclusion

- Repeat finding on a whole genome/whole chromosome basis is possible.
- Suffix trees are a powerful data structure (not only) for repeat finding.
- The flexibility of the data structures allows to find various (related) types of repeats.
- Repeat finding has several applications, some of which are related to repeats only at second glance.