Reducing Squares in Suffix Arrays

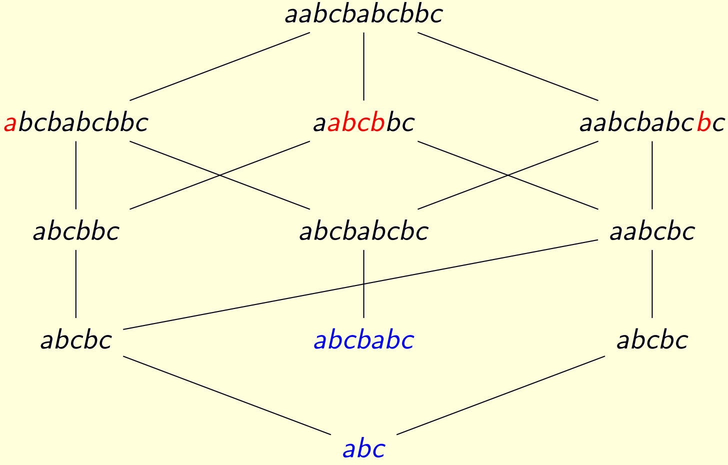
Peter Leupold

University of Leipzig

Prague Stringology Conference 2014

What I Really Want

The duplication history for a string *aabcbabcbbc*.
The direction of reductions is top to bottom:



Why I Want This

- For theoretical reasons (number of normal forms):

$$\mathit{duproots}(n) :=$$

$$\max\{|R| : R \text{ set of all normal forms of a string of length } n\}$$

Why I Want This

- For theoretical reasons (number of normal forms):

$$duproots(n) :=$$

$$\max\{|R| : R \text{ set of all normal forms of a string of length } n\}$$

- Could be useful for compression (Ilie et al.)

Why I Want This

- For theoretical reasons (number of normal forms):

$$\mathit{duproots}(n) :=$$

$$\max\{|R| : R \text{ set of all normal forms of a string of length } n\}$$

- Could be useful for compression (Ilie et al.)
- Use duplication for phylogenetic trees
- ...

Phylogenetic Trees

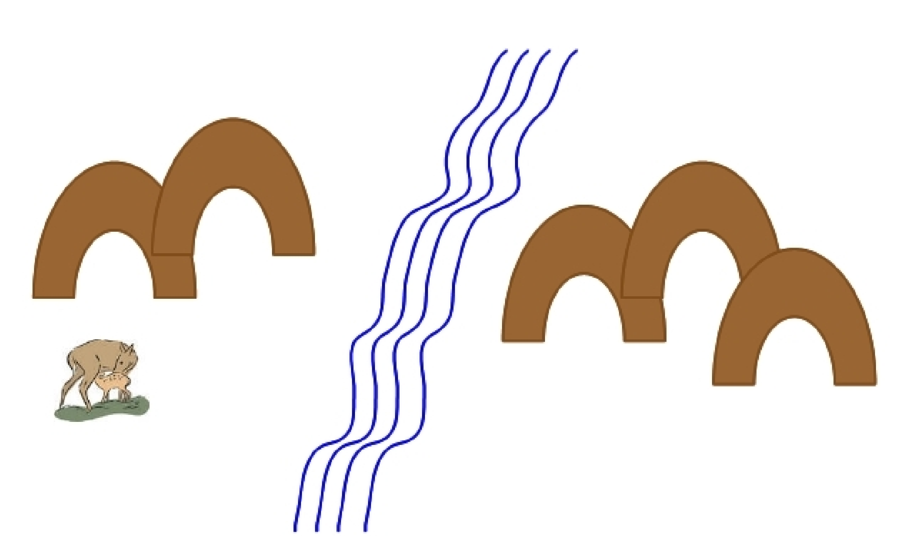
Possible duplication histories interesting for biological investigations.

In the article

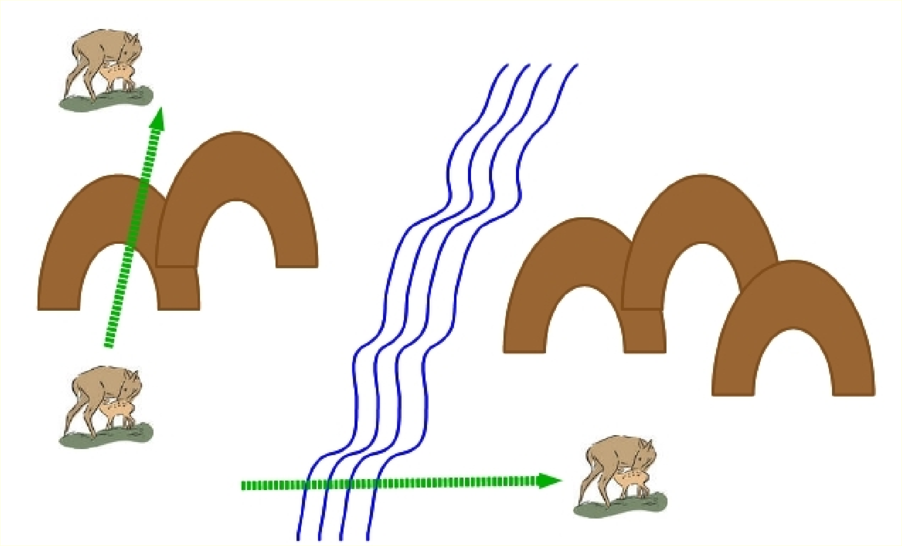
> *Wapinski, I.; Pfeffer, A.; Friedman, N. and Regev, A.:*
> Natural History and Evolutionary Principles of Gene Duplication in Fungi.
> *Nature 449 (2007), pages 54–61.*

the way in which 17 populations of fungi had evolved was induced from looking at the duplication histories of their genomes.

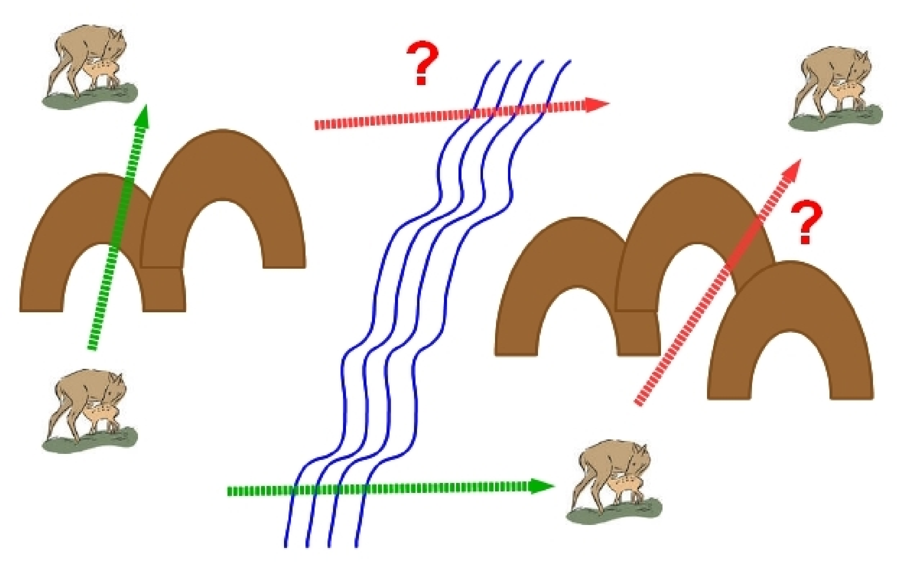
Phylogenetic Trees



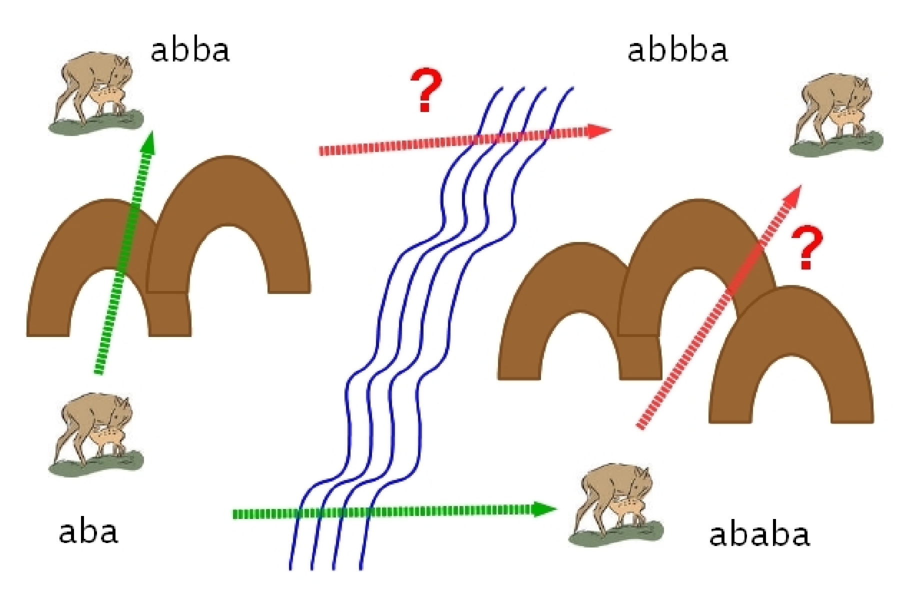
Phylogenetic Trees



Phylogenetic Trees



Phylogenetic Trees



The Big Problem

Theorem (PSC 2009)

For every positive integer ℓ there are words of length ℓ over a four-letter alphabet whose number N of normal forms under eliminating squares is bounded by:

$$\frac{1}{30} 110^{\frac{\ell}{42}} \leq N \leq 2^{\ell}.$$

No efficient algorithm for all cases.

Runs, not Squares

Different square reductions in periodic factor:

abcbcbca   abcbcbca   abcbcbca

↘ ↓ ↙

abcbca

Lemma

Let w be a string with period k. Then any deletion of a factor of length k will lead to the same result.

Naive Computation of all Strings Reachable from w by Reduction of Squares

```
Input: string: w;
Data: stringlist: S (contains w);
while (S nonempty) do
    x := POP(S);
    Construct the suffix array of x;
    if (there are runs in x) then
        foreach run r do
            Reduce one square in r;
            Add new string to S;
        end
    end
    else output x;
end
```

Naive Computation of all Strings Reachable from w by Reduction of Squares

Input: string: w;
Data: stringlist: S (contains w);

```
while (S nonempty) do
    x := POP(S);
    Construct the suffix array of x;
    if (there are runs in x) then
        foreach run r do
            Reduce one square in r;
            Add new string to S;
        end
    end
    else output x;
end
```

Modification of the suffix array by deletion of `bcb` in `abcbbcba`

SA	LCP	
7	1	`a`
0	0	`abcbbcba`
6	1	`ba`
3	1	`bbcba`
4	3	`bcba`
1	0	`bcbbcba`
5	2	`cba`
2		`cbbcba`

$\Rightarrow$

SA	LCP	
$7-3=4$	1	`a`
0	0	`abcba` (new)
$6-3=3$	1	`ba`
		—
$5-3=2$	0	`bcba`
		—
$4-3=1$		`cba`
		—

Modification of the suffix array by deletion of `bcb` in `abcbbcba`

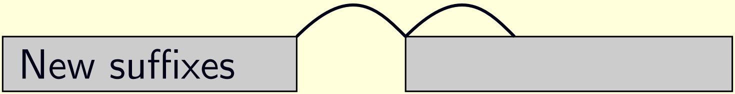
SA	LCP	
7	1	a
0	0	abcbbcba
6	1	ba
3	1	bbcba
4	3	bcba
1	0	bcbbcba
5	2	cba
2		cbbcba

$\Rightarrow$

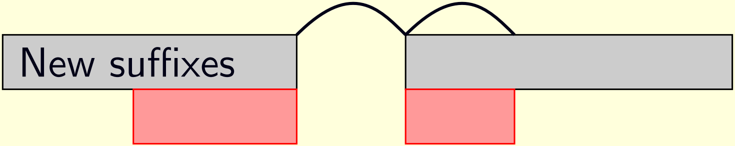
SA	LCP	
$7-3=4$	1	a
0	0	abcba (new)
$6-3=3$	1	ba
		—
$5-3=2$	0	bcba
		—
$4-3=1$		cba
		—

We also need: ISA

The First Small Problem

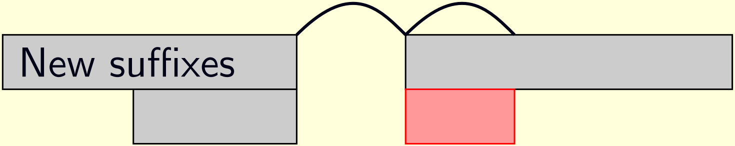


The First Small Problem



Not so new?

The First Small Problem



Not so new?

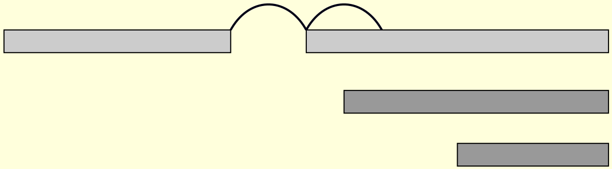
Deletions treated by:

M. Salson, T. Lecroq, M. Léonard, and L. Mouchard:
Dynamic extended suffix arrays.
J. Discrete Algorithms, 8(2) 2010, pp. 241–257.

No Change

Suffixes right of the deletion:

Order and LCP remain the same.

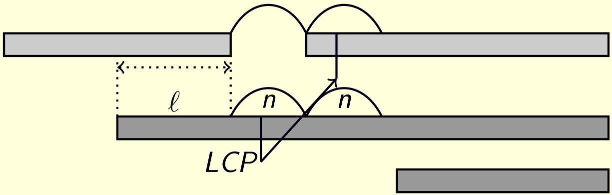


Deleting the left half of the square.

Same Order, Lower LCP

LCP is not greater than $\ell + n$:

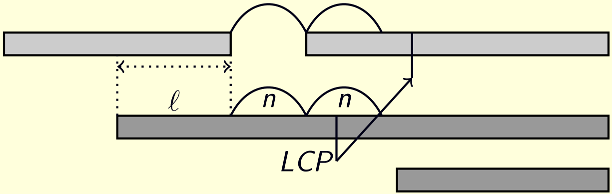
Order unchanged, because $\ell + n$ first letters remain the same.



Order Change

LCP greater than $\ell + n$:

Some letter within the prefix of length LCP might change



Computing the new suffix array

Lemma (Condition for Change in Position)

Let the LCP of two strings z and uvw be k and let $z < uvw$. Then z and $uvvw$ have the same LCP and $z < uvvw$ unless $LCP(z, uvw) \geq |uv|$; in the latter case also $LCP(z, uvvw) \geq |uv|$.

Lemma (No further changes to the left)

Let $LCP[ISA[j]] = k$ in the suffix array of a string w of length $n + 1$. Then for $i < j$ we always have $LCP[ISA[i]] \leq k + j - i$.

Computing the New Suffix Array

Input: string: w; arrays: SA, LCP;
length and pos of square: n,k;

```
for j = n + k to |w| − 1 do
|   SAnew[j]:=SA[j]-n;
end
i := k − 1;
while (LCP[i] > n + k − i AND i ≥ 0) do
|   compute SAnew of w[i ... k − 1]w[k + n ... |w| − 1];
|   compute new LCP[i];
|   /* with methods of Salson et al.             */
|   i := i − 1;
end
for j = 0 to i + m do
|   SAnew[j]:=SA[j];
end
```

Other Small Problems

- Efficient decision whether string has exponentially many ancestors
- Different examples from `abcbabcbc` with several normal forms
- Strategy for traversing the duplication history graph
- Store only changes instead of new suffix array
- Is a different method for run detection better?
- Are there strings over three letters with exponentially many normal forms?

Why I keep thinking about the LCP...