Efficient Seeds Computation Revisited

Michalis Christou, Maxime Crochemore,
Costas S. Iliopoulos, Marcin Kubica,
Solon P. Pissis, **Jakub Radoszewski**,
Wojciech Rytter, Bartosz Szreder, Tomasz Waleń

King's College London & University of Warsaw

CPM Mondello, Palermo, June 29, 2011

Why quasiperiodicity?

Periodicity:

`a b a a a b a a a b a a a b a a`

Why quasiperiodicity?

Periodicity:

`a b a a a b a a a b a a a b a a a b`

Why quasiperiodicity?

Periodicity:

a b a a a b a a a b a a a b a a a b

a b a a b a a b a a a b a a a b

Why quasiperiodicity?

Periodicity:

`a b a a a b a a a b a a a b a a a b`

Quasiperiodicity:

`a b a a b a a b a a a b a a a b`

Types of quasiperiodicity

a b a a b a a b a a a b a a

Types of quasiperiodicity

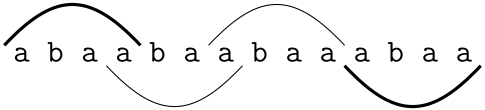
Cover:

a b a a b a a b a a a b a a

every letter of the string is covered by some occurrence of the cover

Types of quasiperiodicity

Cover:



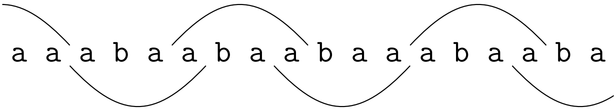
every letter of the string is covered by some occurrence of the cover

Types of quasiperiodicity

a a a b a a b a a b a a a b a a b a

Types of quasiperiodicity

Seed:



every letter of the string is covered by some occurrence of the seed, occurrences may be external

Main related problems

Problem: Cover computation
find the shortest cover (all the covers) of a string u

Main related problems

Problem: Cover computation
find the shortest cover (all the covers) of a string u

Solution:
Apostolico, Farach, and Iliopoulos (1991), Moore and Smyth (1994), $O(n)$ time algorithms.

Main related problems

Problem: Cover computation
find the shortest cover (all the covers) of a string u

Solution:
Apostolico, Farach, and Iliopoulos (1991), Moore and Smyth (1994), $O(n)$ time algorithms.

Harder problem: Cover array
compute $\mathrm{C}[1..n]$, where $\mathrm{C}[i]$ is the shortest cover of the string $u[1..i]$

Main related problems

Problem: Cover computation
find the shortest cover (all the covers) of a string u

Solution:
Apostolico, Farach, and Iliopoulos (1991), Moore and Smyth (1994), $O(n)$ time algorithms.

Harder problem: Cover array
compute $\mathrm{C}[1..n]$, where $\mathrm{C}[i]$ is the shortest cover of the string $u[1..i]$

Solution:
Breslauer (1992), $O(n)$ time algorithm.

Main related problems

Problem: Cover computation
find the shortest cover (all the covers) of a string u

Solution:
Apostolico, Farach, and Iliopoulos (1991), Moore and Smyth (1994), $O(n)$ time algorithms.

Harder problem: Cover array
compute $C[1..n]$, where $C[i]$ is the shortest cover of the string $u[1..i]$

Solution:
Breslauer (1992), $O(n)$ time algorithm.

Another problem: Seed computation
find the shortest seed (all the seeds) of a string

Main related problems

Problem: Cover computation
find the shortest cover (all the covers) of a string u

Solution:
Apostolico, Farach, and Iliopoulos (1991), Moore and Smyth (1994), $O(n)$ time algorithms.

Harder problem: Cover array
compute $\mathrm{C}[1..n]$, where $\mathrm{C}[i]$ is the shortest cover of the string $u[1..i]$

Solution:
Breslauer (1992), $O(n)$ time algorithm.

Another problem: Seed computation
find the shortest seed (all the seeds) of a string

Solution:
Iliopoulos, Moore & Park (1996), $O(n \log n)$ time algorithm.

Main contributions

1. Left seeds
We introduce a natural intermediate notion between seeds and covers and give $O(n)$ time algorithms for computing the shortest left seed and the left seed array.

2. Seed array
We show how to compute the seed array in $O(n^2)$ time.

3. New (simpler) seeds computation
We present a novel approach to seed computation. Our algorithm works in $o(n \log n)$ time for some cases.

Left/right seeds

Cover:

a b a a b a a b a a a b a a

Seed:

a a a b a a b a a b a a a b a a b a

Left/right seeds

Cover:

a b a a b a a b a a a b a a

Seed:

a a a b a a b a a b a a a b a a b a

a b a a b a a b a a a b a a b a

Left/right seeds

Cover:

a b a a b a a b a a a b a a

Seed:

a a a b a a b a a b a a a b a a b a

Left seed:

a b a a b a a b a a a b a a b a

is a prefix of the string, however its occurrence
may exceed the right end of the string

Left/right seeds

Cover:

a b a a b a a b a a a b a a

Seed:

a a a b a a b a a b a a a b a a b a

a a b a a b a a b a a a b a a

Left/right seeds

Cover:

a b a a b a a b a a a b a a

Seed:

a a a b a a b a a b a a a b a a b a

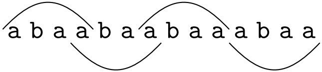
Right seed:

a a b a a b a a b a a a b a a

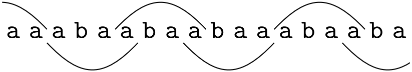
is a suffix of the string, however its occurrence may exceed the left end of the string

Left/right seeds

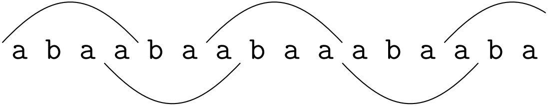
Cover:



Seed:



Left seed:



is a prefix of the string, however its occurrence may exceed the right end of the string

Left seeds computation

Problem: Left seed computation
find the shortest lest seed of a string u

Left seeds computation

Problem: Left seed computation
find the shortest lest seed of a string u

Harder problem: Left seed array
compute $\mathrm{LSeed}[1..n]$, where $\mathrm{LSeed}[i]$ is the shortest left seed of the string $u[1..i]$

Left seeds computation

Problem: Left seed computation
find the shortest lest seed of a string u

Harder problem: Left seed array
compute $\mathrm{LSeed}[1\,.\,.\,n]$, where $\mathrm{LSeed}[i]$ is the shortest left seed of the string $u[1\,.\,.\,i]$

Solution:
We present $O(n)$ time algorithms solving both the problems.

Left seeds computation

The period of a string

We say that a positive integer p is the (shortest) *period* of a string $u = u_1 \dots u_n$ (notation: $p = \mathrm{per}(u)$) if p is the smallest positive number, such that $u_i = u_{i+p}$, for $i = 1, \dots, n-p$.

Left seeds computation

The period of a string

We say that a positive integer p is the (shortest) *period* of a string $u = u_1 \dots u_n$ (notation: $p = \mathrm{per}(u)$) if p is the smallest positive number, such that $u_i = u_{i+p}$, for $i = 1, \dots, n-p$.

Lemma. The length of the shortest left seed of u equals:

$$\min\{\mathsf{C}[j] \ : \ \mathrm{per}(u) \le j \le |u|\}$$

where $\mathsf{C}[1 \mathinner{.\,.} n]$ is the cover array of u.

Left seeds computation

The period of a string

We say that a positive integer p is the (shortest) *period* of a string $u = u_1 \ldots u_n$ (notation: $p = \mathrm{per}(u)$) if p is the smallest positive number, such that $u_i = u_{i+p}$, for $i = 1, \ldots, n-p$.

Lemma. The length of the shortest left seed of u equals:

$$\min\{\mathrm{C}[j] \ : \ \mathrm{per}(u) \le j \le |u|\}$$

where $\mathrm{C}[1\,..\,n]$ is the cover array of u.

Corollary. The left seed of a string can be computed in $O(n)$ time.

The proof

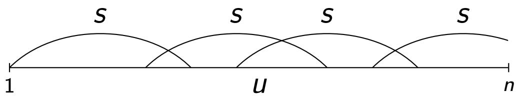
Lemma. The length of the shortest left seed of u equals:

$$\min\{C[j] \ : \ \mathrm{per}(u) \le j \le |u|\}$$

where $C[1..n]$ is the cover array of u.

Proof.

Assume that s is a left seed of u.



The proof

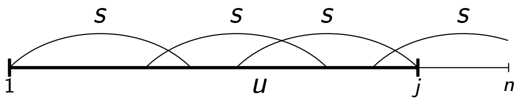
Lemma. The length of the shortest left seed of u equals:

$$\min\{C[j] \ : \ \mathrm{per}(u) \leq j \leq |u|\}$$

where $C[1..n]$ is the cover array of u.

Proof.

Assume that s is a left seed of u.



Then s is a cover of $u[1..j]$ for some j.

The proof

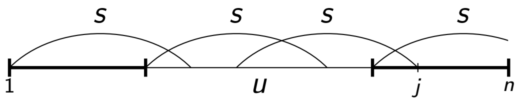
Lemma. The length of the shortest left seed of u equals:

$$\min\{C[j] \;:\; \mathrm{per}(u) \le j \le |u|\}$$

where $C[1..n]$ is the cover array of u.

Proof.

Assume that s is a left seed of u.



Then s is a cover of $u[1..j]$ for some j.

The string u has a border $\ge n - j$, hence $\mathrm{per}(u) \le j$.

The proof

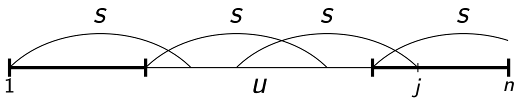
Lemma. The length of the shortest left seed of u equals:

$$\min\{C[j] \ : \ \text{per}(u) \le j \le |u|\}$$

where $C[1..n]$ is the cover array of u.

Proof.

Assume that s is a left seed of u.



Then s is a cover of $u[1..j]$ for some j.

The string u has a border $\ge n - j$, hence $\text{per}(u) \le j$.

(Recall that $\text{per}(u) + \text{border}(u) = |u|$).

The proof

Lemma. The length of the shortest left seed of u equals:

$$\min\{\mathrm{C}[j] \;:\; \mathrm{per}(u) \leq j \leq |u|\}$$

where $\mathrm{C}[1..n]$ is the cover array of u.

Proof (cont).

We have proved that the shortest left seed of u corresponds to one of the covers $\mathrm{C}[j]$ for $j \geq \mathrm{per}(u)$.

We need to show that *each value* $\mathrm{C}[j]$ for $j \geq \mathrm{per}(u)$ corresponds to some left seed of u.

The proof

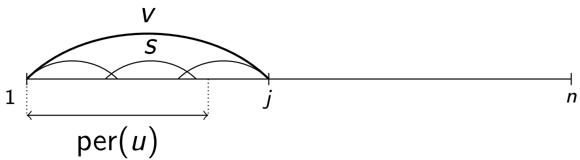
Lemma. The length of the shortest left seed of u equals:

$$\min\{C[j] \;:\; \mathrm{per}(u) \le j \le |u|\}$$

where $C[1..n]$ is the cover array of u.

Proof (cont).

Assume that s is a cover of $v = u[1..j]$ for some $j \ge \mathrm{per}(u)$.



The proof

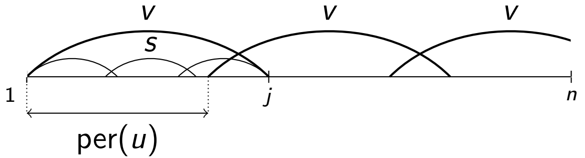
Lemma. The length of the shortest left seed of u equals:

$$\min\{C[j] \;:\; \mathrm{per}(u) \le j \le |u|\}$$

where $C[1..n]$ is the cover array of u.

Proof (cont).

Assume that s is a cover of $v = u[1..j]$ for some $j \ge \mathrm{per}(u)$.



Then v is a left seed of u.

The proof

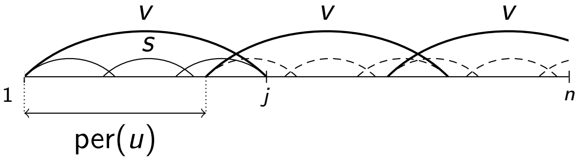
Lemma. The length of the shortest left seed of u equals:

$$\min\{C[j] \ : \ \mathrm{per}(u) \leq j \leq |u|\}$$

where $C[1..n]$ is the cover array of u.

Proof (cont).

Assume that s is a cover of $v = u[1..j]$ for some $j \geq \mathrm{per}(u)$.



Then v is a left seed of u.

Hence, s is also a left seed of u.

Left seeds computation

Lemma. The length of the shortest left seed of u equals:

$$\min\{\mathsf{C}[j] \;:\; \mathsf{per}(u) \le j \le |u|\}$$

where $\mathsf{C}[1\,..\,n]$ is the cover array of u.

Left seeds computation

Lemma. The length of the shortest left seed of u equals:

$$\min\{\mathsf{C}[j] \ :\ \mathrm{per}(u) \le j \le |u|\}$$

where $\mathsf{C}[1\,.\,.\,n]$ is the cover array of u.

Corollary 2. The left seed array can be computed as follows:

$$\mathsf{LSeed}[i] = \min\{\mathsf{C}[j] \ :\ \mathsf{P}[i] \le j \le i\}$$

where $\mathsf{P}[1\,.\,.\,n]$ is the period array

Left seeds computation

Lemma. The length of the shortest left seed of u equals:

$$\min\{\mathsf{C}[j] \ : \ \mathsf{per}(u) \le j \le |u|\}$$

where $\mathsf{C}[1..n]$ is the cover array of u.

Corollary 2. The left seed array can be computed as follows:

$$\mathsf{LSeed}[i] = \min\{\mathsf{C}[j] \ : \ \mathsf{P}[i] \le j \le i\}$$

where $\mathsf{P}[1..n]$ is the period array (recall that it can be computed in $O(n)$ time).

Left seeds computation

Lemma. The length of the shortest left seed of u equals:
$$\min\{\mathsf{C}[j] \ : \ \mathrm{per}(u) \le j \le |u|\}$$
where $\mathsf{C}[1\mathinner{.\,.} n]$ is the cover array of u.

Corollary 2. The left seed array can be computed as follows:
$$\mathsf{LSeed}[i] = \min\{\mathsf{C}[j] \ : \ \mathsf{P}[i] \le j \le i\}$$
where $\mathsf{P}[1\mathinner{.\,.} n]$ is the period array (recall that it can be computed in $O(n)$ time).

The problem reduces to RMQ on the C array — $O(n)$ time algorithm.

Seed array

Problem: Seed array
compute Seed$[1..n]$, where Seed$[i]$ is the shortest seed of the string $u[1..i]$

Seed array

Problem: Seed array
compute Seed$[1..n]$, where Seed$[i]$ is the shortest seed of the string $u[1..i]$

A naive method yields $O(n^2 \log n)$ time.

Seed array

Problem: Seed array
compute Seed$[1..n]$, where Seed$[i]$ is the shortest seed of the string $u[1..i]$

A naive method yields $O(n^2 \log n)$ time.

We present an $O(n^2)$ time algorithm.

The solution

ALGORITHM SeedArray(u)

Seed[1] := 1;
for $i := 2$ **to** n **do**
    Seed[i] := Seed[$i - 1$];
    while $u[1..i]$ does not have a seed of length Seed[i] **do**
        Seed[i] := Seed[i] + 1;
return Seed[$1..n$];

The solution

ALGORITHM SeedArray(u)

```
Seed[1] := 1;
for i := 2 to n do
    Seed[i] := Seed[i − 1];
    while u[1..i] does not have a seed of length Seed[i] do
        Seed[i] := Seed[i] + 1;
return Seed[1..n];
```

We develop an $O(n)$ time test:

SeedsOfAGivenLength(u, k)

which checks if u has a seed of length k.

The solution

ALGORITHM SeedArray(u)

Seed$[1] := 1$;
for $i := 2$ **to** n **do**
    Seed$[i] :=$ Seed$[i-1]$;
    while $u[1..i]$ does not have a seed of length Seed$[i]$ **do**
        Seed$[i] :=$ Seed$[i] + 1$;
return Seed$[1..n]$;

We develop an $O(n)$ time test:

SeedsOfAGivenLength(u, k)

which checks if u has a seed of length k.

(This test uses the suffix arrays of u.)

Seed computation

We present a new $O(n \log n)$ time algorithm for seed computation.

It can be used to check if u has the shortest seed of length $\geq m$ in $O(n \log(n/m))$ time. Hence, finding the shortest seed of length $m = \Theta(n)$ can be done in $O(n)$ time.

Links between the notions

Seed:

`a a a b a a b a a b a a a b a a b a`

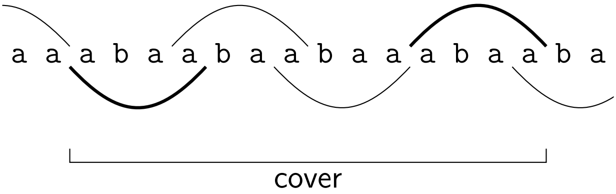
Links between the notions

Seed:

a a a b a a b a a b a a a b a a b a

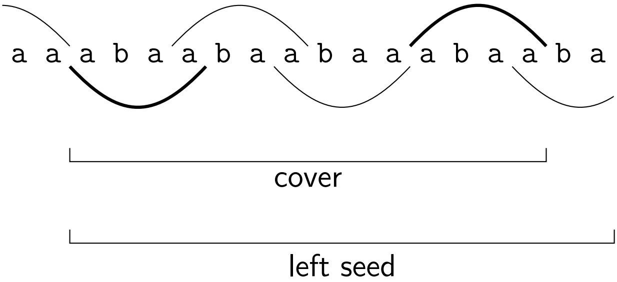
Links between the notions

Seed:



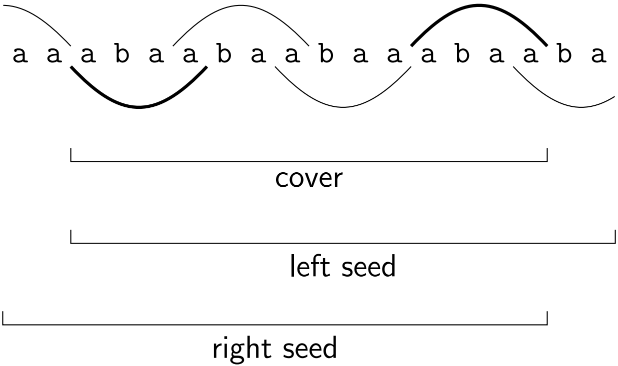
Links between the notions

Seed:



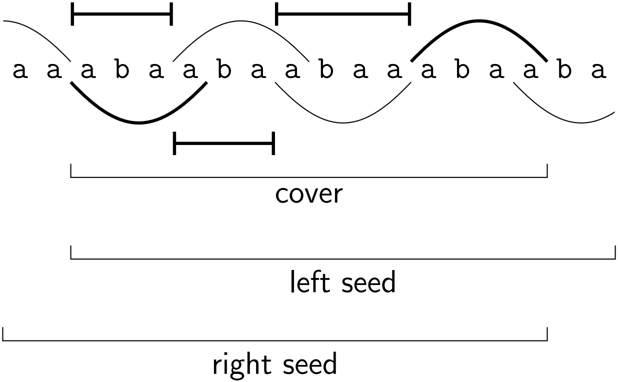
Links between the notions

Seed:



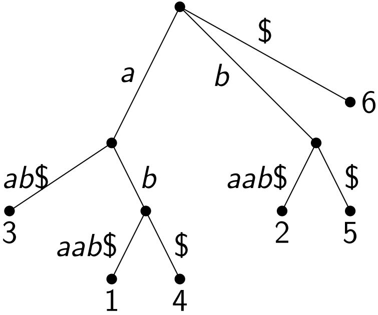
Links between the notions

Seed:



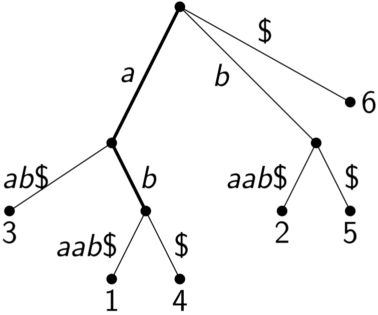
The string s is a cover of $u[i\,.\,.\,j]$ if $\mathrm{maxgap}(s) \leq |s|$.

The suffix tree



It suffices to compute maxgaps for all the explicit nodes of the suffix tree. E.g., $\text{maxgap}(ab) = \text{maxgap}(\{1, 4\}) = 3$, $\text{maxgap}(a) = \text{maxgap}(\{1, 3, 4\}) = 2$.

The suffix tree



It suffices to compute maxgaps for all the explicit nodes of the suffix tree. E.g., $\text{maxgap}(ab) = \text{maxgap}(\{1, 4\}) = 3$, $\text{maxgap}(a) = \text{maxgap}(\{1, 3, 4\}) = 2$.

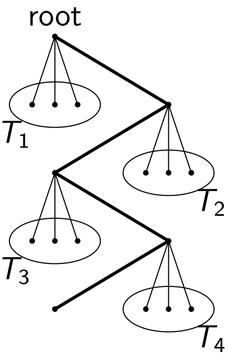
It is easier to compute prefix maxgaps (i.e., the maxima of the maxgap values in the path from a node to the root).

Prefix maxgaps

We show how to compute all the prefix maxgaps in a path down the suffix tree in linear time.

Prefix maxgaps

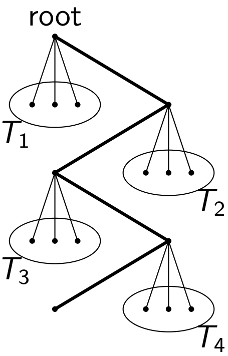
We show how to compute all the prefix maxgaps in a path down the suffix tree in linear time.



We obtain a recursive algorithm.

Prefix maxgaps

We show how to compute all the prefix maxgaps in a path down the suffix tree in linear time.

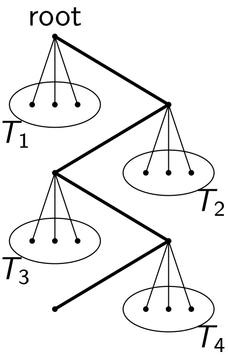


We obtain a recursive algorithm.

Each time we choose the heaviest path in the suffix tree. We obtain $O(\log n)$ levels of recursion and $O(n \log n)$ total time.

Prefix maxgaps

We show how to compute all the prefix maxgaps in a path down the suffix tree in linear time.



If we search for the shortest seed of length $\geq m$ then it suffices to consider several subtrees of the tree, each of size $O(n/m)$. We obtain an $O(n \log(n/m))$ time algorithm.

Summary

Cover	$O(n)$
Cover array	$O(n)$
Left seed	$\mathbf{O(n)}$
Left seed array	$\mathbf{O(n)}$
Seed	$O(n \log n)$ $[\ \mathbf{O(n \log(n/m))}\ ]$
Seed array	$\mathbf{O(n^2)}$

Summary

Cover	$O(n)$
Cover array	$O(n)$
Left seed	$\mathbf{O(n)}$
Left seed array	$\mathbf{O(n)}$
Seed	$O(n \log n)$ [$\mathbf{O(n \log(n/m))}$]
Seed array	$\mathbf{O(n^2)}$

Thank you for your attention!