2D Dictionary Matching in Small Space

Shoshana Neuburger

Graduate Center of CUNY

2/29/2012

Dissertation Defense

Outline

Thesis Accomplishments:

1. New Techniques
2. Succinct 2D Dictionary Matching with No Slowdown
3. Dynamic Succinct 2D Dictionary Matching
4. Software for Succinct 1D Dictionary Matching

Problem Definition

2D Dictionary Matching

Input:

- Dictionary $D = \{P_1, P_2, \ldots, P_d\}$ of pattern matrices
- Text matrix T

Output:

- (h, i, j) such that pattern P_h occurs at location (i, j) in T
  $T[i+k, j+l] = P_h[k+1, l+1]$

Small-Space

Challenge:

- Limited storage capacity in devices.
- Massive Proliferation of Data

Goal: efficient algorithms with respect to both time and space .

Small-Space

Challenge:

- Limited storage capacity in devices.
- Massive Proliferation of Data

Goal: efficient algorithms with respect to both time and space .

Hon et al. (2011): Time-space optimal 1D dictionary matching.

This work: first to focus on 2D dictionary matching in small space.

Small-Space 2D

2D linear-time single pattern matching
Crochemore et al. (1995):

- Preprocessing: linear time within log space.
- Text Scanning: linear time, $O(1)$ extra space.

Use small-space 2D single pattern matching for set of patterns

* requires several scans of text.

2D Dictionary Matching

Existing 2D dictionary matching algorithms:

- Bird (1977) / Baker (1978)
- Amir, Farach (1992)
- Giancarlo (1993)
- Idury, Schaffer (1993)

Require working space proportional to dictionary size.

2D Dictionary Matching

Bird / Baker

- Convert 2D data to 1D representation.
- Name patterns rows.
- Name text positions.
- Use 1D dictionary matching to find pattern occurrences.

2D Dictionary Matching

Bird / Baker

- Convert 2D data to 1D representation.
- Name patterns rows.
- Name text positions.
- Use 1D dictionary matching to find pattern occurrences.

Text is processed once!

Our work: succinct version of Bird/Baker algorithm.

Bird /Baker Algorithm

Pattern

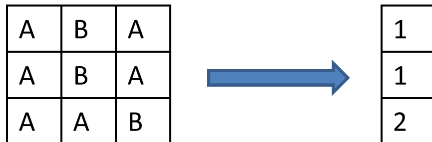
A	B	A
A	B	A
A	A	B

Text

A	A	B	A	B	A	A
B	A	B	A	B	A	B
A	A	B	A	A	B	B
B	A	A	B	A	A	A
A	B	A	B	A	A	A
B	B	A	A	B	A	B
B	B	B	A	B	A	B

Bird /Baker Algorithm

Pattern Preprocessing



Bird /Baker Algorithm

Text Scanning

A	A	B	A	B	A	A
B	A	B	A	B	A	B
A	A	B	A	A	B	B
B	A	A	B	A	A	A
A	B	A	B	A	A	A
B	B	A	A	B	A	B
B	B	B	A	B	A	B

0	0	2	1	0	1	0
0	0	0	1	0	1	0
0	0	2	1	0	2	0
0	0	0	2	1	0	0
0	0	1	0	1	0	0
0	0	0	0	2	1	0
0	0	0	0	0	1	0

Bird /Baker Algorithm

Text Scanning

A	A	B	A	B	A	A
B	A	B	A	B	A	B
A	A	B	A	A	B	B
B	A	A	B	A	A	A
A	B	A	B	A	A	A
B	B	A	A	B	A	B
B	B	B	A	B	A	B

0	0	2	1	0	1	0
0	0	0	1	0	1	0
0	0	2	1	0	2	0
0	0	0	2	1	0	0
0	0	1	0	1	0	0
0	0	0	0	2	1	0
0	0	0	0	0	1	0

Bird /Baker Algorithm

Text Scanning

A	A	B	A	B	A	A
B	A	B	A	B	A	B
A	A	B	A	A	B	B
B	A	A	B	A	A	A
A	B	A	B	A	A	A
B	B	A	A	B	A	B
B	B	B	A	B	A	B

0	0	2	1	0	1	0
0	0	0	1	0	1	0
0	0	2	1	0	2	0
0	0	0	2	1	0	0
0	0	1	0	1	0	0
0	0	0	0	2	1	0
0	0	0	0	0	1	0

Problem Definition

2D Dictionary Matching

Input:

- Dictionary of d patterns, each is $m \times m$ in size.
- Text T of size $n \times n$.

Output:

- All positions in text at which a dictionary pattern occurs.

Preprocessing Space

Bird and Baker:

- Aho-Corasick automaton of pattern rows.
- $O(dm^2 \log dm^2)$ extra bits of preprocessing space.

New technique:

- Groups pattern rows into equivalence classes.
- $O(dm \log dm)$ extra bits of preprocessing space.

Text Scanning Space

Bird and Baker:

- Process entire text at once.
- $O(n^2 \log \, dm)$ bits of space to label text.

New technique:

- Small overlapping text blocks of size $3m/2 \times 3m/2$.
- $O(m^2 \log dm)$ bits of space to label text.

Working space is independent of text size.

Our Method

Overview of Algorithm:

- Name pattern rows to form 1D dictionary.
- Name each text block row.
- 1D dictionary matching to locate candidates .
- Verify candidates to find pattern occurrences .

Repeat for each overlapping text block of size $3m/2 \times 3m/2$.

1D Periodicity

Definition

A string p is *periodic* in u if $p = u^k u'$ where u' is a proper prefix of u, u is primitive, and $k \geq 2$.

$$aabcaabcaabcaa$$

1D Periodicity

Definition

A string p is *periodic* in u if $p = u^k u'$ where u' is a proper prefix of u, u is primitive, and $k \geq 2$.

$$aabcaabcaabcaa$$
$$aabcaabcaabcaa$$

1D Periodicity

Definition

A string p is *periodic* in u if $p = u^k u'$ where u' is a proper prefix of u, u is primitive, and $k \geq 2$.

We divide patterns into 2 groups based on 1D periodicity.

In each case, different difficulties to overcome.

Types of Patterns

Case I:

Patterns with ALL rows periodic, period $\leq m/4$.
Problem: can have more candidates than the space we allow.

Case II:

Patterns contain aperiodic row or row with period $> m/4$.
Problem: several patterns can overlap in both directions.

Types of Patterns

Case I:

Patterns with ALL rows periodic, period $\leq m/4$.
Problem: can have more candidates than the space we allow.

New techniques:

* Lyndon word naming
* Witness tree
* 2D Lyndon words

Lyndon Words

Definition

Two words x, y are conjugate if $x = uv$, $y = vu$ for some u, v.

Definition

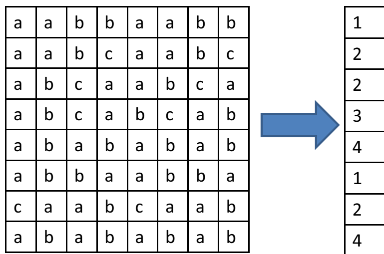
A Lyndon word is a primitive string which is lexicographically smaller than any of its conjugates.

Canonization computes the least conjugate of a word.

Naming

New technique for naming rows:

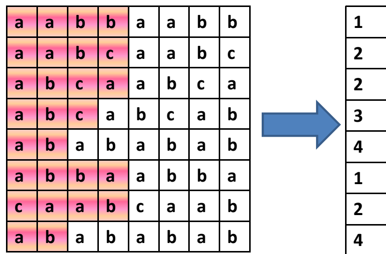
same name if periods are conjugate



Naming

New technique for naming rows:

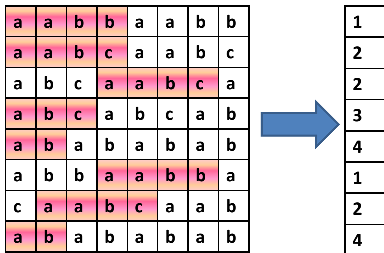
same name if periods are conjugate



Naming

New technique for naming rows:

same name if periods are conjugate



Naming

New technique for naming rows:

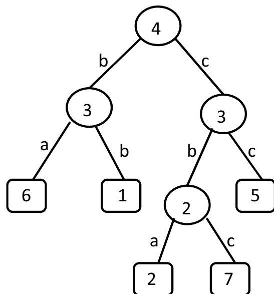
same name if periods are conjugate

How is this done in linear time, yet small space? witness tree

- Witness tree stores a distinction between two names.
- To name a new row, it is compared to only one other row.

Witness Tree

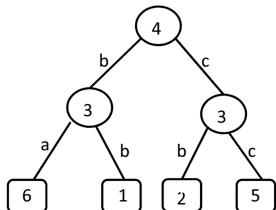
Witness tree for Lyndon words of length 4:



Name	Period size	Lyndon word
1	4	aabb
2	4	aabc
3	3	abc
4	2	ab
5	4	aacc
6	4	aaab
7	4	acbc

Witness Tree

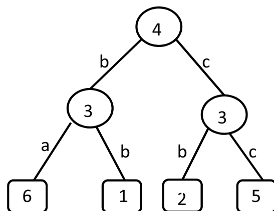
To give acbc name 7 and append to witness tree:



Name	Period size	Lyndon word
1	4	aabb
2	4	aabc
3	3	abc
4	2	ab
5	4	aacc
6	4	aaab
7	4	acbc

Witness Tree

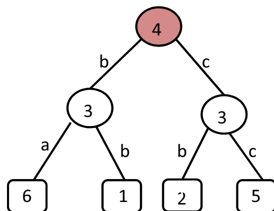
To give acbc name 7 and append to witness tree:



Name	Period size	Lyndon word
1	4	aabb
2	4	aabc
3	3	abc
4	2	ab
5	4	aacc
6	4	aaab
7	4	acbc

Witness Tree

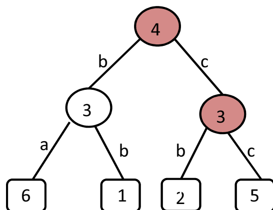
To give acbc name 7 and append to witness tree:



Name	Period size	Lyndon word
1	4	a a b b
2	4	a a b c
3	3	a b c
4	2	a b
5	4	a a c c
6	4	a a a b
7	4	a c b **c**

Witness Tree

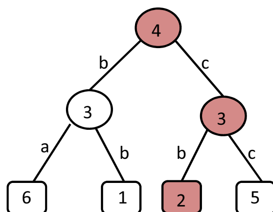
To give acbc name 7 and append to witness tree:



Name	Period size	Lyndon word
1	4	a a b b
2	4	a a b c
3	3	a b c
4	2	a b
5	4	a a c c
6	4	a a a b
7	4	a c **b** c

Witness Tree

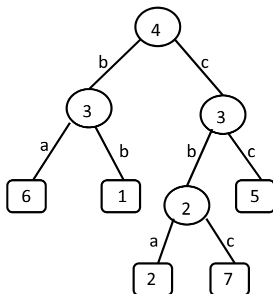
To give acbc name 7 and append to witness tree:



Name	Period size	Lyndon word
1	4	a a b b
2	4	a a b c
3	3	a b c
4	2	a b
5	4	a a c c
6	4	a a a b
7	4	a c b c

Witness Tree

To give acbc name 7 and append to witness tree:



Name	Period size	Lyndon word
1	4	aabb
2	4	aabc
3	3	abc
4	2	ab
5	4	aacc
6	4	aaab
7	4	acbc

Preprocess 1D Patterns

1. Linearize 2D patterns in dictionary.
2. Construct AC automaton of 1D patterns.
3. Compute LCM table of each 1D pattern.
4. Compute 2D Lyndon word of each 1D pattern.

Preprocess 1D Patterns

1. Linearize 2D patterns in dictionary.
2. Construct AC automaton of 1D patterns.
3. Compute LCM table of each 1D pattern.
4. Compute 2D Lyndon word of each 1D pattern.

LCM

Why store the Least Common Multiple (LCM) of 1D patterns?

- Text can have more pattern occurrences than space we allow.
- However, they occur at regular intervals.
- Summarized by occurrence's left and right endpoints + LCM of pattern.

Pattern Preprocessing

Summary of pattern preprocessing:

1. For each pattern row,
   1. compute period and canonize
   2. name row
   3. store period size, name, first Lyndon word occurrence ($LYpos$).
2. Construct AC automaton of 1D patterns.
3. Compute LCM table for each 1D pattern.
4. For multiple patterns of same 1D name, build offset tree.

Time: $O(dm^2)$
Extra Space: $O(dm \log dm)$ bits

Text Scanning

Text scanning stage:

1. Name rows of text block.
2. Identify candidates.
3. Verify candidates.

Naming Text

Lemma

At most one maximal periodic substring of length $\geq m$ with period $\leq m/4$ can occur in a text block row of size $3m/2$.

Process each text block row:

- Name text block rows same way as pattern rows.
- Find maximal periodic substring around midpoint.
- Each text block row receives only one name.

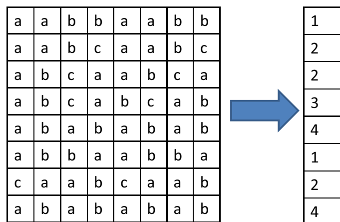
Text Scanning

Text scanning stage:

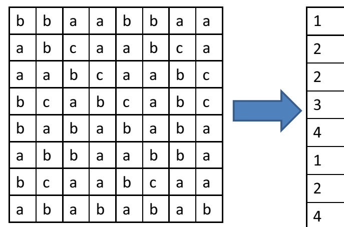
1. For each text block row,
   1. compute period and canonize
   2. name row
   3. store period size, name, first Lyndon word occurrence (*LYpos*).
2. Identify candidates: 1D dictionary matching.
3. Verify candidates.

Verification

Pattern 1



Pattern 2



Same 1D name but different 2D patterns!

h-periodicity

Definition

A 2D $m \times m$ pattern is *h-periodic*, or horizontally periodic, if two copies of the pattern can be aligned in the top row so that there is no mismatch in the region of overlap and the number of overlapping columns is $\geq m/2$.

h-periodicity

Definition

A 2D $m \times m$ pattern is *h-periodic*, or horizontally periodic, if two copies of the pattern can be aligned in the top row so that there is no mismatch in the region of overlap and the number of overlapping columns is $\geq m/2$.

a	*a*	*b*	*c*	*a*	*a*	*b*	*c*	*a*
a	a	b	b	a	a	b	b	a
a	b	c	a	a	b	c	a	a
c	c	c	c	c	c	c	c	c
a	b	a	b	a	b	a	b	a
a	b	b	a	a	b	b	a	a
c	a	a	b	c	a	a	b	c
a	b	a	b	a	b	a	b	a

h-periodicity

aabcaabcaabcaa
aabcaabcaabcaa

a	*a*	*b*	*c*
a	a	b	b
a	b	c	a
c	c	c	c
a	b	a	b
a	b	b	a
c	a	a	b
a	b	a	b

a	*a*	*b*	*c*	*a*
a	a	b	b	a
a	b	c	a	a
c	c	c	c	c
a	b	a	b	a
a	b	b	a	a
c	a	a	b	c
a	b	a	b	a

a	*b*	*c*	*a*
a	b	b	a
b	c	a	a
c	c	c	c
b	a	b	a
b	b	a	a
a	a	b	c
b	a	b	a

h-periodicity

Definition

A 2D $m \times m$ pattern is *h-periodic*, or horizontally periodic, if two copies of the pattern can be aligned in the top row so that there is no mismatch in the region of overlap and the number of overlapping columns is $\geq m/2$.

Definition

The *h-period* of an h-periodic pattern is the minimum column number at which the pattern can be aligned over itself.

Verification

Horizontally Consistent Patterns

- Same 1D representation

  and

- Can occur at overlapping positions on text block row

b	b	a	a	b	b	a	a	b	b
a	b	c	a	a	b	c	a	a	b
a	a	b	c	a	a	b	c	a	a
b	c	a	b	c	a	b	c	a	b
b	a	b	a	b	a	b	a	b	a
a	b	b	a	a	b	b	a	a	b
b	c	a	a	b	c	a	a	b	c
a	b	a	b	a	b	a	b	a	b

Verification

Horizontally Consistent Patterns

- Same 1D representation

  and

- Can occur at overlapping positions on text block row

Lemma

Two h-periodic patterns with the same 1D representation are horizontally consistent iff the LYpos of all their rows are shifted by C mod period size of the row, where C is an integer.

Horizontal Consistency

Lemma

Two h-periodic patterns with the same 1D representation are horizontally consistent iff the LYpos of all their rows are shifted by C mod period size of the row, where C is an integer.

		x	x	x	x		
			x	x	x	x	
x	x	x	x				
		x	x	x			
	x	x					
			x	x	x	x	
		x	x	x	x		
x	x						

x	x	x	x				
	x	x	x	x			
		x	x	x	x		
x	x	x					
	x	x					
	x	x	x	x			
x	x	x	x				
x	x						

$C = 2$

Verification

Single pass verification:

- Group horizontally consistent patterns together.
- Compute 2D Lyndon word and classify patterns.
- Store 2D Lyndon words in offset tree .
- Compute 2D Lyndon word of text and traverse tree.

Horizontal 2D Conjugacy

Definition

P_1 and P_2, are *horizontal 2D conjugate* if $P_1 = UV$, $P_2 = VU$ for some horizontal prefix U and horizontal suffix V of P_1.

If the h-periods of two patterns are horizontal 2D conjugate, then the 2D patterns are horizontally consistent.

2D Lyndon word

Each conjugate of an h-period has a distinct *LYpos* sequence.

Definition

The *2D Lyndon word* of a matrix is the *LYpos* array that is the smallest over all the horizontal 2D conjugates of the matrix, for the numerical ordering.

2D Lyndon word

Pattern 2

b	b	a	a	b	b	a	a
a	b	c	a	a	b	c	a
a	a	b	c	a	a	b	c
b	c	a	b	c	a	b	c
b	a	b	a	b	a	b	a
a	b	b	a	a	b	b	a
b	c	a	a	b	c	a	a
a	b	a	b	a	b	a	b

LYpos
2
3
0
2
1
3
2
0

2D Lyndon word

Horizontal 2D conjugate

b	a	a	b	b	a	a	b
b	c	a	a	b	c	a	a
a	b	c	a	a	b	c	a
c	a	b	c	a	b	c	a
a	b	a	b	a	b	a	b
b	b	a	a	b	b	a	a
c	a	a	b	c	a	a	b
b	a	b	a	b	a	b	a

LYpos
1
2
3
1
0
2
1
1

$$C = 1$$

2D Lyndon word

2D Lyndon Word

a	a	b	b	a	a	b	b
c	a	a	b	c	a	a	b
b	c	a	a	b	c	a	a
a	b	c	a	b	c	a	b
b	a	b	a	b	a	b	a
b	a	a	b	b	a	a	b
a	a	b	c	a	a	b	c
a	b	a	b	a	b	a	b

LYpos
0
1
2
0
1
1
0
0

$C = 2$

Computing 2D Lyndon word

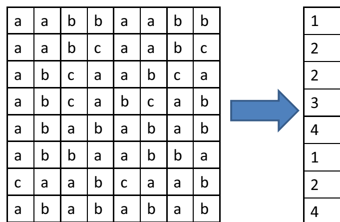
Each conjugate of an h-period has a distinct *LYpos* sequence.

How many horizontal 2D conjugates does a pattern have? LCM !

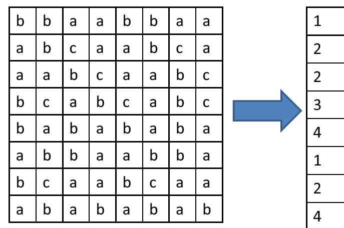
Compute 2D LYndon word for h-period of $m \times m$ matrix in
$O(m)$ time and
$O(m \log m)$ bits of extra space.

Verification

Pattern 1



Pattern 2



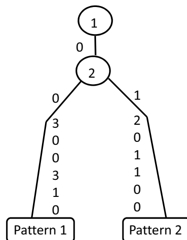
Not horizontally consistent.

Offset Tree

Pattern1 and Pattern2:

- same 1D name
- h-periods not horizontal 2D conjugate

Offset Tree



Name	Period size	Pattern1 LYpos	Pattern1 Lyndon Word	Pattern2 Lypos	Pattern2 Lyndon Word
1	4	0	0	2	0
2	4	0	0	3	1
2	4	3	3	0	2
3	3	0	0	2	0
4	2	0	0	1	1
1	4	3	3	3	1
2	4	1	1	2	0
4	2	0	0	0	0

Text Scanning

Text scanning stage:

1. For each text block row,
   1. compute period and canonize
   2. name row
   3. store period size, name, first Lyndon word occurrence ($LYpos$).
2. Identify candidates: 1D dictionary matching.
3. Verify candidates on each text row: offset tree.

Time: $O(n^2)$
Extra Space: $O(m \log dm)$ bits

Summary of Case I

Algorithm for patterns with highly periodic rows:

- Name pattern rows to form 1D dictionary.
- Name each text block row.
- Use 1D dictionary matching to find candidates .
- Verify candidates to find pattern occurrences .

Repeat for each overlapping text block of size $3m/2 \times 3m/2$.

Types of Patterns

Case II:

Patterns contain aperiodic row or row with period $> m/4$.
Problem: several patterns can overlap in both directions.

New techniques:

* Dynamic dueling
* Witness tree

Types of Patterns

Case II:

Patterns contain aperiodic row or row with period $> m/4$.
Problem: several patterns can overlap in both directions.

- Many 1D names can overlap in a text block row.
- Identification of candidates is simpler.
- Identify candidates with aperiodic row of each pattern.
- Difficulty: single pass verification.

Pattern Preprocessing

Pattern Preprocessing:

1. Construct (compressed) AC automaton of first aperiodic row of each pattern.
   Store row number of each row within pattern.
2. Form a compressed AC automaton of the pattern rows.
3. Name pattern rows.
   Index 1D patterns of names in suffix tree.
4. Construct witness tree of pattern rows.
   Preprocess for LCA.

Time: $O(dm^2)$
Extra Space: $O(dm \log m)$ bits

Searching Text

Text Scanning:

1. Identify candidates.
2. Eliminate inconsistent candidates.
3. Verify pattern occurrences.

Searching Text

Step 1: Identify candidates

- 1D dictionary matching of a non-periodic row of each pattern.
- $O(dm)$ candidates in a text block.
- Possibly several candidates at a single text position.

Searching Text

Text Scanning:

1. Identify candidates.
2. Eliminate inconsistent candidates.
3. Verify pattern occurrences.

Searching Text

Step 2: Eliminate inconsistent candidates in each column

Two candidates are consistent if all positions of overlap match.

Vertically consistent candidates:

- In the same column.
- Suffix/prefix match in 1D representations.

Overlapping segments of consistent candidates can be verified simultaneously $\Rightarrow$ single pass verification.

Searching Text

Step 2: Eliminate inconsistent candidates in each column
How to eliminate inconsistent candidates? duels .

Dueling for 2D *single* pattern matching [Amir et al. (1994)]

* Store witness for all conflicting overlaps.
* No witness $\Rightarrow$ consistent candidates.
* Duel: compare text location to witness, kill 1+ candidates.

Dictionary matching: candidates for *different* patterns.
Too many witnesses to store? Dynamic dueling generates.

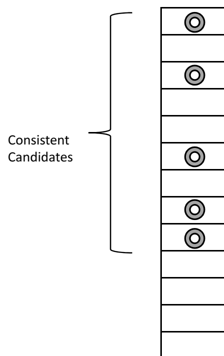
Searching Text

Step 2: Eliminate inconsistent candidates in each column

- Duels from top to bottom of rows.
- Consistency is transitive.
- Duel between vertically inconsistent candidates.

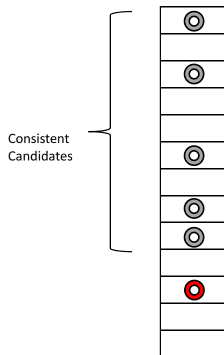
Searching Text

Step 2: Eliminate inconsistent candidates in each column



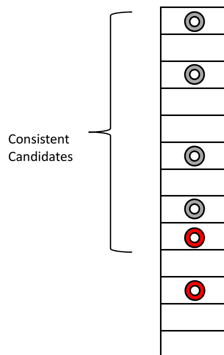
Searching Text

Step 2: Eliminate inconsistent candidates in each column



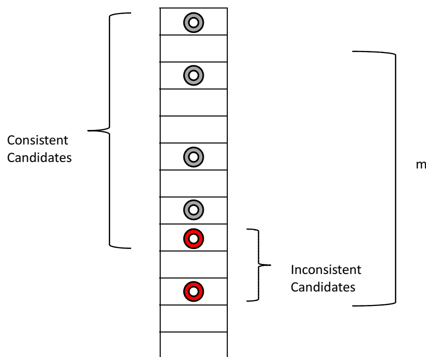
Searching Text

Step 2: Eliminate inconsistent candidates in each column



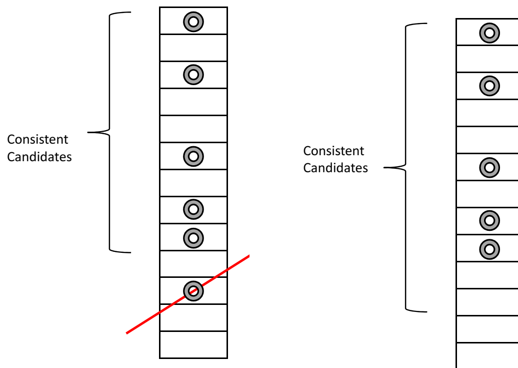
Searching Text

Step 2: Eliminate inconsistent candidates in each column



Searching Text

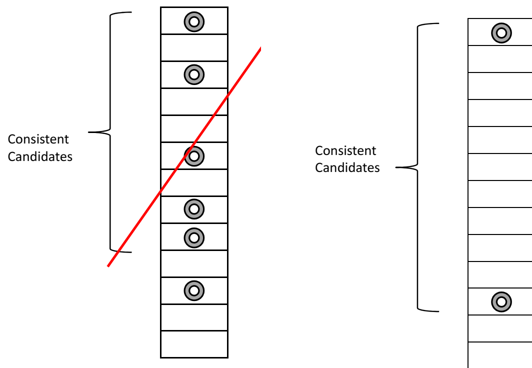
Step 2: Eliminate inconsistent candidates in each column



If last candidate wins duel

Searching Text

Step 2: Eliminate inconsistent candidates in each column

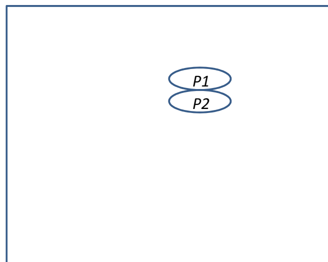


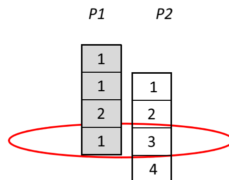
If new candidate wins duel

Searching Text

Step 2: Eliminate inconsistent candidates in each column

How to duel between candidates?





Searching Text

Step 2: Eliminate inconsistent candidates in each column

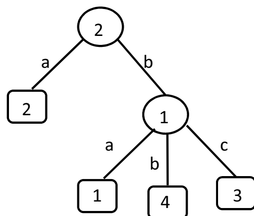
How to duel between candidates?

1. Use 1D representation, named pattern rows.
   Compute LCP of suffixes to find a row-witness .
2. Generate witness between row names.
   LCA query in witness tree.

Searching Text

Step 2: Eliminate inconsistent candidates in each column

How to generate witness between row names?



Name	String
1	abbb
2	aabc
3	cbca
4	bbaa

Searching Text

Text Scanning:

1. Identify candidates.
2. Eliminate inconsistent candidates.
3. Verify pattern occurrences.

Searching Text

Step 3: Verify pattern occurrences.

- Limited to vertically consistent candidates.
- Single scan of text block.
- Process one row at a time.
- Mark text positions that expect pattern row.
- Verify with compressed AC automaton of pattern rows.

Searching Text

Text Scanning:

1. Identify candidates.
2. Eliminate inconsistent candidates.
3. Verify pattern occurrences.

Time: $O(n^2)$
Extra Space: $O(dm \log dm)$ bits

Summary of Case II

Algorithm for patterns with aperiodic row:

- Self-index of dictionary in entropy-compressed space.
- Name pattern rows to form 1D dictionary.
- Use 1D dictionary matching to find candidates .
- Eliminate inconsistent candidates.
- Scan text once to find pattern occurrences .

Repeat for each overlapping text block of size $3m/2 \times 3m/2$.

Compressed Matching

Patterns and text are all in compressed form.

Definition

An algorithm is strongly inplace if the extra space it uses is proportional to the optimal compression of the data.

Key property of LZ78:

can sequentially decompress using constant space in time linear in the uncompressed string.

Compressed Matching

Definition

An algorithm is strongly inplace if the extra space it uses is proportional to the optimal compression of the data.

- Amir, Landau, Sokol (2003): strongly-inplace single pattern matching in 2D LZ78-compressed data.
- Consider row-by-row linearization of 2D data.
- Text scanning time is linear in size of uncompressed data.

Cannot access complete dictionary when processing text.

Compressed Matching

Definition

An algorithm is strongly inplace if the extra space it uses is proportional to the optimal compression of the data.

Our algorithm for patterns with highly periodic rows is linear time and strongly inplace .

Amir, Landau, Sokol (2003): $O(m^3)$ preprocessing time.

Our techniques improve their algorithm to linear $O(m^2)$.

Outline

Thesis Accomplishments:

1. New Techniques
2. Succinct 2D Dictionary Matching with No Slowdown
3. Dynamic Succinct 2D Dictionary Matching
4. Software for Succinct 1D Dictionary Matching

Dynamic Dictionary

Sahinalp and Vishkin (1996):
Dynamic 1D dictionary matching

- Linear time and space
- Not automaton based
- Naming technique
- Can replace AC in Bird / Baker $\Rightarrow$ linear time and space dynamic 2D dictionary matching.

Dynamic Dictionary

Our Contribution:

Succinct dynamic 2D dictionary matching.

- Adapts to changes in dictionary
  - * Efficiently insert pattern.
  - * Efficiently delete pattern.
  - * Without reprocessing entire dictionary.
- Modification of our techniques for static dictionary.
- Dynamic succinct version of Bird/Baker algorithm.
- Uses dynamic data structures:
  - dynamic compressed suffix tree
  - Sahinalp and Vishkin's dynamic 1D dictionary matching

Outline

Thesis Accomplishments:

1. New Techniques
2. Succinct 2D Dictionary Matching with No Slowdown
3. Dynamic Succinct 2D Dictionary Matching
4. Software for Succinct 1D Dictionary Matching

Small-Space 1D

1D dictionary matching in small space :

Space (bits)	Search Time	Reference
$O(\ell \log \ell)$	$O(n + occ)$	Aho-Corasick (1975)
$O(\ell)$	$O((n + occ) \log^2 \ell)$	Chan et al. (2007)
$\ell H_k(D) + o(\ell \log \sigma) + O(d \log \ell)$	$O(n(\log^\epsilon l + \log d) + occ)$	Hon et al. (2008)
$\ell(H_0 + O(1)) + O(d \log(\ell/d))$	$O(n + occ)$	Belazzougui (2010)
$\ell H_k(D) + O(\ell)$	$O(n + occ)$	Hon et al. (2010)

d is the number of patterns in D.
ℓ is the total size of the dictionary.

These theoretical contributions have not been implemented.

1D Dictionary Matching

For 1D data,
Time-Space efficient dictionary matching has been achieved.

* Only in the theoretical realm.
* Rely on complex data structures
  that have *not* been implemented.

Our Contribution:
Efficient software for succinct dictionary matching
that relies on popular succinct data structures.

Software Development

Creation of our succinct 1D dictionary matching program:

1. Coded Ukkonen's suffix tree construction algorithm.
2. Modified suffix tree to form generalized suffix tree.
3. Wrote program to perform dictionary matching over generalized suffix tree.
4. Ported dictionary matching code to use compressed suffix tree.

Compressed Suffix Tree

Compressed suffix tree (CST)

- Compressed self-index.
- Replaces input data and answers queries.
- No more space than underlying data.
- Minor slowdown in compressed suffix array as well.

Compressed Suffix Tree

Uncompressed Suffix Tree: $O(n \log n)$ bits of space.

Compressed Suffix Tree:

1. Sadakane (2007):
   $O(n \log \sigma)$ bits, $O(polylog(n))$ slowdown.
2. Russo et al. (2008):
   kth order empirical entropy, $O(\log n)$ slowdown.
3. Fischer et al. (2009):
   kth order empirical entropy, sub-logarithmic slowdown.
4. Ohlebusch, Fischer, Gog (2011):
   some queries in $O(1)$ time.

Have all been implemented.

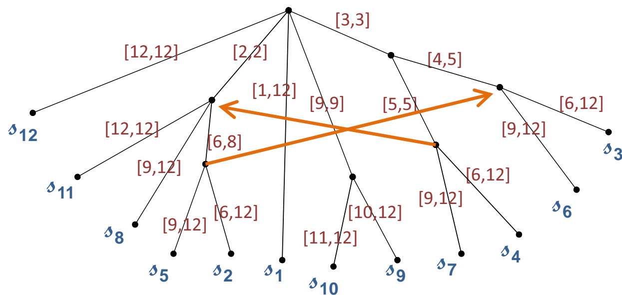
Suffix Links

Definition

A suffix link is a pointer from an internal node labeled xS to another internal node labeled S, where x is an arbitrary character and S is a possibly empty substring.

Suffix links facilitate traversal of suffix tree *during* and *after* construction.

Suffix Links



Software

Algorithm for 1D dictionary matching on suffix tree:

- Generalized suffix tree: index of several strings.
- Ukkonen can insert one string at a time.
- Our algorithm: modeled after Ukkonen's suffix tree construction algorithm.
    - Online processing of text.
    - Linear time: as if inserting new pattern.
    - Skip-count trick uses suffix links.

Software

Algorithm for 1D dictionary matching on suffix tree:

- Generalized suffix tree: index of several strings.
- Ukkonen can insert one string at a time.
- Our algorithm: modeled after Ukkonen's suffix tree construction algorithm.
  - Online processing of text.
  - Linear time: as if inserting new pattern.
  - Skip-count trick uses suffix links.

Small-Space: compressed suffix tree.

Software

Software for succinct 1D dictionary matching

- Uses SDSL compressed suffix tree
- Space meets entropy-compressed bounds
- Linear time, with slowdown to query compressed self-index

Future Work

Small space 2D dictionary matching variations:

- Square patterns of different sizes
- Rectangular patterns of different sizes
- Approximate matching
  - mismatches
  - insertions
  - deletions
  - swap
- Software for dynamic succinct dictionary matching
- Software for succinct 2D dictionary matching

Thank you to the examining committee!

* Prof. Dina Sokol
* Prof. Amihood Amir
* Prof. Amotz Bar-Noy
* Prof. Stathis Zachos