Succinct 2D Dictionary Matching with No Slowdown

Shoshana Neuburger
and
Dina Sokol

City University of New York

Problem Definition

Dictionary Matching

Input:

- Dictionary $D = P_1, P_2, \ldots, P_d$ containing d patterns.
- Text T of length n.

Output:

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

Applications

Dictionary Matching

- Search for specific phrases in a book
- Scanning file for virus signatures
- Network intrusion detection systems
- Searching DNA sequence for a set of motifs

Small-Space 1D

In many devices, storage capacity is limited.

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

Small-Space 1D

In many devices, storage capacity is limited.

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

1D single pattern matching in linear time and $O(1)$ working space:

- Galil and Seiferas (1981)
- Crochemore and Perrin (1991)
- Rytter (2003)

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)

2D Dictionary Matching

Existing 2D dictionary matching algorithms:

- Bird (1977) / Baker (1978)
- Amir, Farach (1992)
- 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: mimic Bird/Baker algorithm in small space.

Bird /Baker Algorithm

Pattern

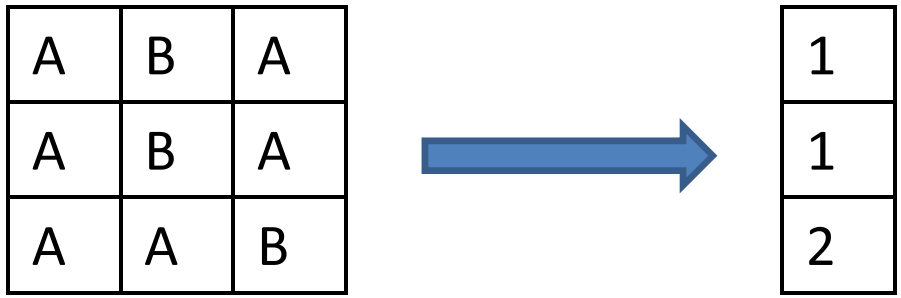
A	B	A
A	B	A
A	A	B

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

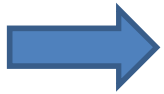
Pattern Preprocessing



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

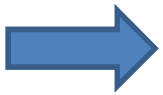


0	0	2	1	0	1	0
0	0	0	1	0	1	0
0	0	2	1	0	2	0
0	0	0	2	1	0	0
0	0	1	0	1	0	0
0	0	0	0	2	1	0
0	0	0	0	0	1	0

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 algorithm:

- Compressed Aho-Corasick automaton of pattern rows.
- 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.

To save space

- 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

Compressed AC automaton [Hon et al. (2010)]:

- Separates the three functions of the AC automaton.
- Encodes each function differently.
- Space complexity meets $H_k(D)$, kth order empirical entropy.

Black-box replacement for AC automata in Bird / Baker algorithm.

Dictionary Size

Using *compressed* AC automata in small blocks of text

Theorem

If $d > m$, we can solve the 2D dictionary matching problem in linear $O(dm^2 + n^2)$ time and $\ell H_k(D) + O(\ell) + O(dm \log dm)$ bits of space.

ℓ is the number of states in the AC automaton of pattern rows

Dictionary Size

Using *compressed* AC automata in small blocks of text

Theorem

If $d > m$, we can solve the 2D dictionary matching problem in linear $O(dm^2 + n^2)$ time and $\ell H_k(D) + O(\ell) + O(dm \log dm)$ bits of space.

ℓ is the number of states in the AC automaton of pattern rows

We focus on case when $d < m$.

1D Periodicity

Definition

A string p is *periodic* in u if $p = u'u^k$ where u' is a suffix 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 in which all pattern rows are 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 in which all pattern rows are periodic, period $\leq m/4$.
Problem: can have more candidates than the space we allow.

Algorithm published in CPM 2010 for compressed data.

Use *conjugacy* of periods to group similar pattern rows in the same equivalence class.

Types of Patterns

Case I:

Patterns in which all pattern rows are periodic, period $\leq m/4$.
Problem: can have more candidates than the space we allow.

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 $3m/2$.

Types of Patterns

Case II:

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

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.
- Difficult to verify in single pass over text.

Pattern Preprocessing

Pattern Preprocessing:

1. Construct (compressed) AC automaton of first aperiodic row of each pattern.
   Store row number of each of these rows within the patterns.
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.

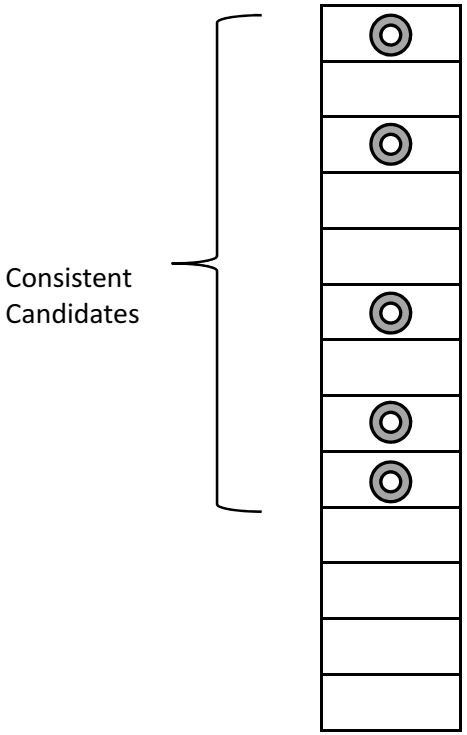
Searching Text

Step 2: Eliminate inconsistent candidates in each column

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

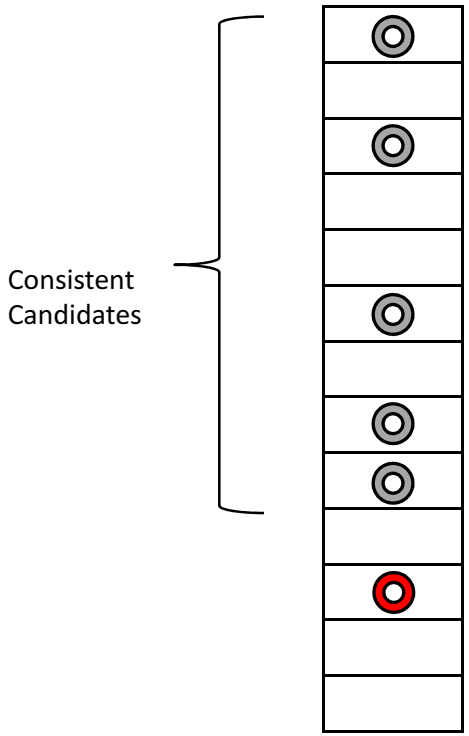
Searching Text

Step 2: Eliminate inconsistent candidates in each column



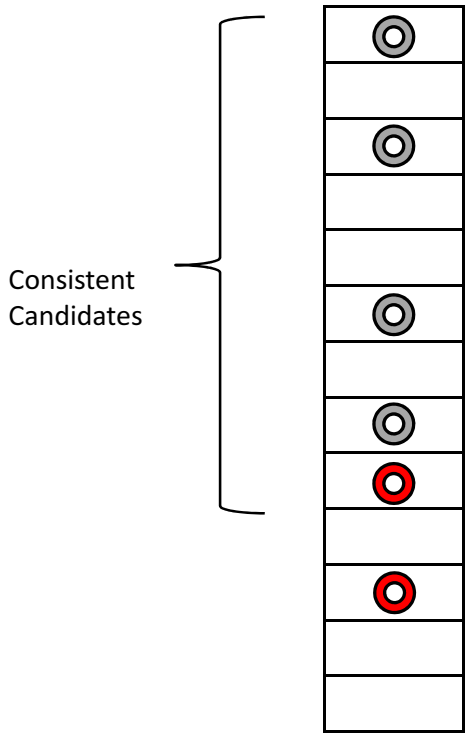
Searching Text

Step 2: Eliminate inconsistent candidates in each column



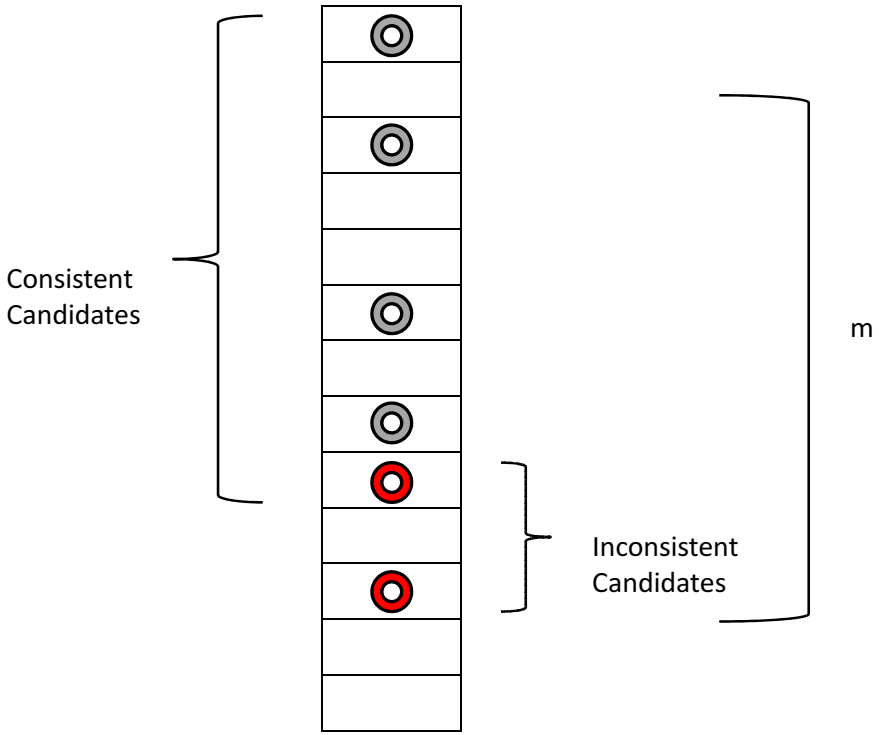
Searching Text

Step 2: Eliminate inconsistent candidates in each column



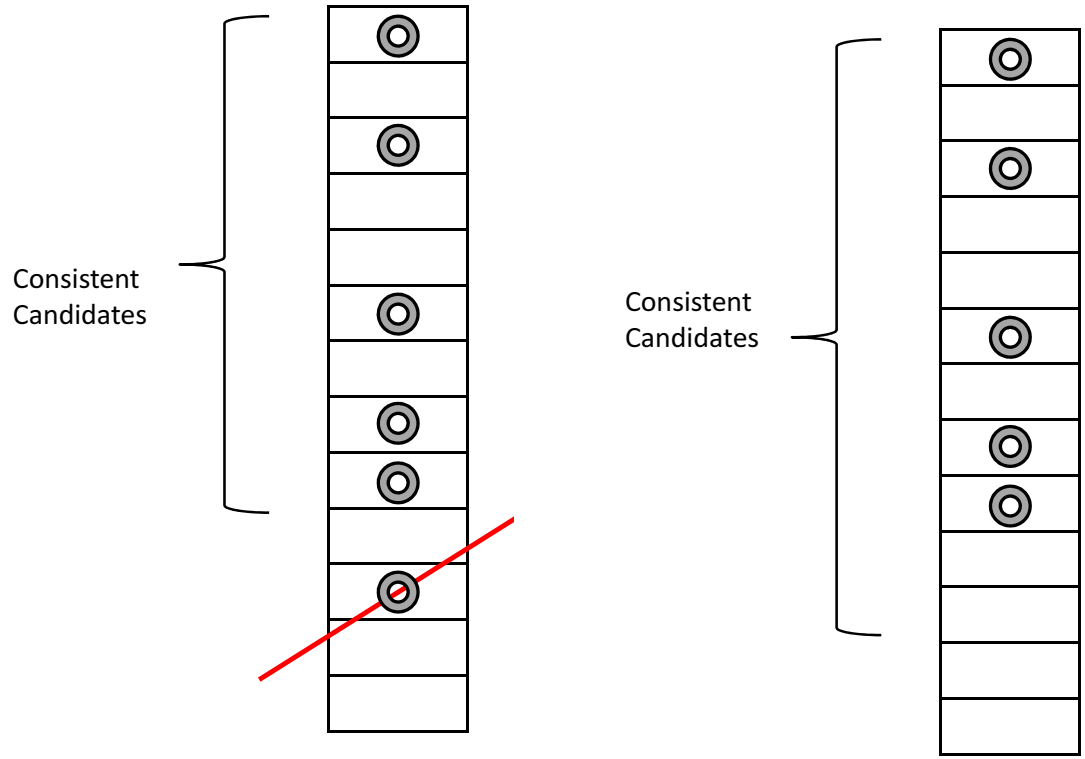
Searching Text

Step 2: Eliminate inconsistent candidates in each column



Searching Text

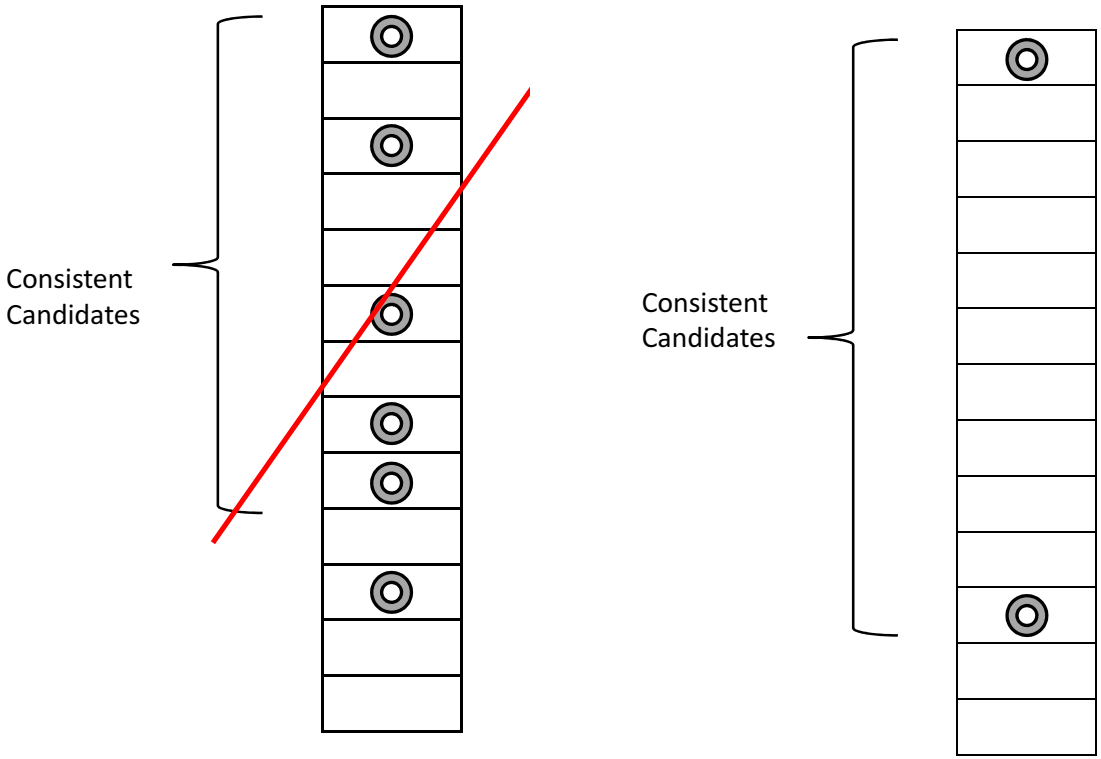
Step 2: Eliminate inconsistent candidates in each column



If last candidate wins duel

Searching Text

Step 2: Eliminate inconsistent candidates in each column

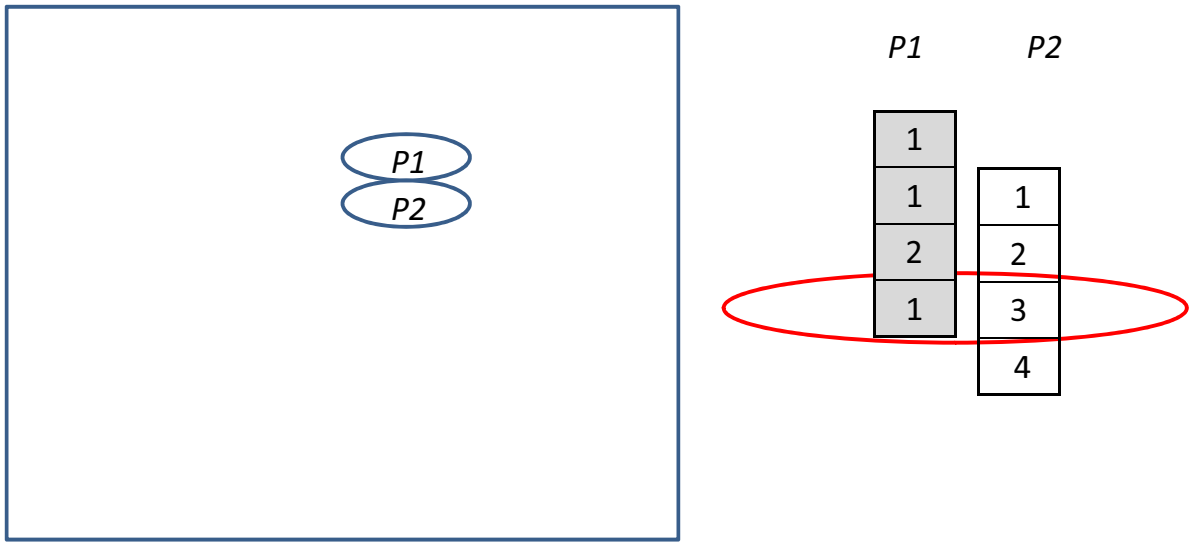


If new candidate wins duel

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?

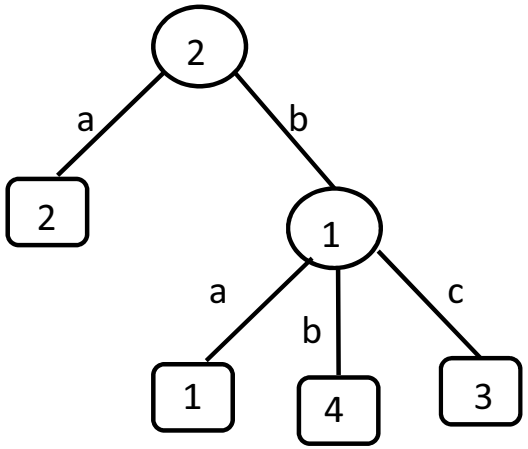
1. Consider 1D representation of pattern names.
   Compute LCP of suffixes to find a row-witness .
2. Generate witness between row names.
   LCA query in witness tree.

Searching Text

Step 2: Eliminate inconsistent candidates in each column

How to generate witness?

Witness Tree



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

Summary

New approach to 2D dictionary matching in small-space:

- Time complexity is linear.
- Preprocess dictionary in time proportional to dictionary size.
- Store dictionary in entropy-compressed space.
- Scan text in time proportional to text, independent of dictionary size.
- Overall, $O(dm \log dm)$ extra bits of space.

Thank you!