Self-Indexed Text Compression using Straight-Line Programs

Francisco Claude[1] and Gonzalo Navarro[2]

[1]David R. Cheriton School of Computer Science, University of Waterloo
[2]Department of Computer Science, University of Chile

August 5, 2009

Contents

Outline

Motivation

- Grammar-based compression aims to represent a sequence building a small grammar the derives the sequence
- Can be used to compress collections with one single grammar (not covered in this work)
- We focus on the special case of Straight-Line Programs (SLPs)

Motivation

- Much research has been done for searching for a pattern in a grammar-compressed text (linear in the number of symbols in the grammar)
- Many compression algorithms can be mapped to an SLP (LZ78, LZ77*, Re-Pair)
- Our goal is to build an indexing method for SLPs in order to support faster pattern matching, using little space

Basic Concepts

Straigh-Line Program

A *Straight-Line Program (SLP)* $\mathcal{G} = (X = \{X_1, \ldots, X_n\}, \Sigma)$ is a grammar that defines a single finite sequence $T[1, u]$, drawn from an alphabet $\Sigma = [1, \sigma]$ of terminals. It has n rules, which must be of the following types:

- $X_i \rightarrow \alpha$, where $\alpha \in \Sigma$. It represents string $\mathcal{F}(X_i) = \alpha$.
- $X_i \rightarrow X_\ell X_r$, where $\ell, r < i$. It represents string $\mathcal{F}(X_i) = \mathcal{F}(X_\ell)\mathcal{F}(X_r)$.

We call $\mathcal{F}(X_i)$ the *phrase generated* by nonterminal X_i, and $T = \mathcal{F}(X_n)$.

Basic Concepts

Height of a symbol

The *height* of a symbol X_i in the SLP $\mathcal{G} = (X, \Sigma)$ is defined as

$height(X_i) = 1$ if $X_i \rightarrow \alpha \in \Sigma$, and
$height(X_i) = 1 + \max(height(X_\ell), height(X_r))$ if $X_i \rightarrow X_\ell X_r$.

The height of the SLP is $height(\mathcal{G}) = height(X_n)$.

Example (Re-Pair based)

alabaralaalabarda

Example (Re-Pair based)

alabaralaalabarda

Example (Re-Pair based)

alabaralaalabarda

- $A \longrightarrow al$

Example (Re-Pair based)

AabarAaAabarda

- $A \longrightarrow al$

Example (Re-Pair based)

AabarAaAabarda

- $A \rightarrow al$

Example (Re-Pair based)

AabarAaAabarda

- $A \rightarrow al$
- $B \rightarrow Aa$

Example (Re-Pair based)

BbarBBbarda

- $A \rightarrow al$
- $B \rightarrow Aa$

Example (Re-Pair based)

BbarBBbarda

- $A \rightarrow al$
- $B \rightarrow Aa$

Example (Re-Pair based)

BbarBBbarda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$

Example (Re-Pair based)

BbCBBbCda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$

Example (Re-Pair based)

BbCBBbCda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$

Example (Re-Pair based)

BbCBBbCda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$

Example (Re-Pair based)

BDBBDda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$

Example (Re-Pair based)

BDBBDda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$

Example (Re-Pair based)

BDBBDda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$
- $E \rightarrow BD$

Example (Re-Pair based)

EBEda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$
- $E \rightarrow BD$

Example (Re-Pair based)

EBEda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$
- $E \rightarrow BD$

Example (Re-Pair based)

FEda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$
- $E \rightarrow BD$
- $F \rightarrow EB$

Example (Re-Pair based)

Gda

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$
- $E \rightarrow BD$
- $F \rightarrow EB$
- $G \rightarrow FE$

Example (Re-Pair based)

Ha

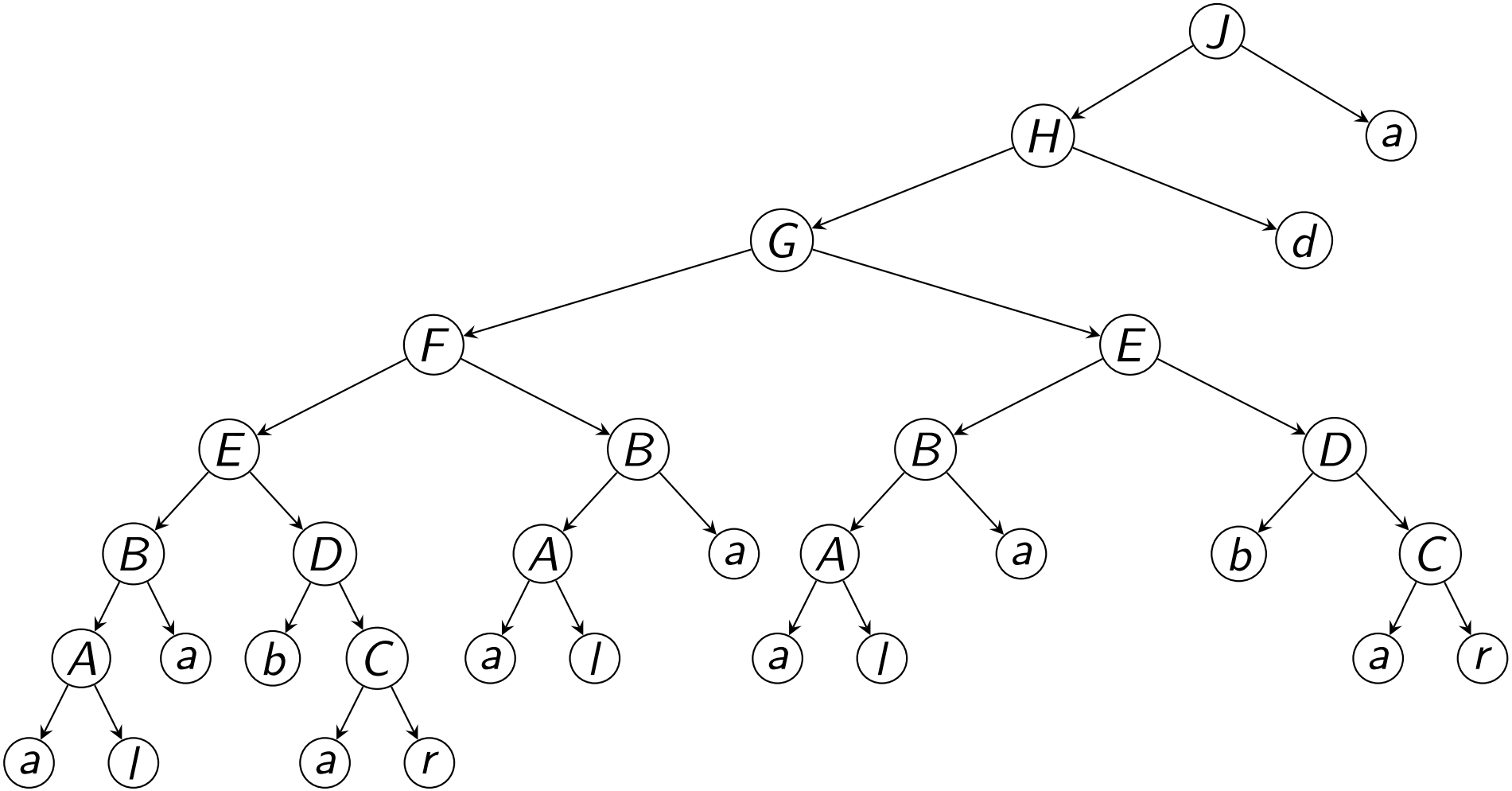
- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$
- $E \rightarrow BD$
- $F \rightarrow EB$
- $G \rightarrow FE$
- $H \rightarrow Gd$

Example (Re-Pair based)

J

- $A \rightarrow al$
- $B \rightarrow Aa$
- $C \rightarrow ar$
- $D \rightarrow bC$
- $E \rightarrow BD$
- $F \rightarrow EB$
- $G \rightarrow FE$
- $H \rightarrow Gd$
- $J \rightarrow Ha$

Example (Re-Pair based)



Rank/Select Operations

Rank/Select

For a sequence S of length n, drawn from an alphabet Σ or size σ:

- $rank(c, i)$: number of occurrences of symbol c in $S[1 \dots i]$.
- $select(c, j)$: position of the j-th occurrence of c in S.

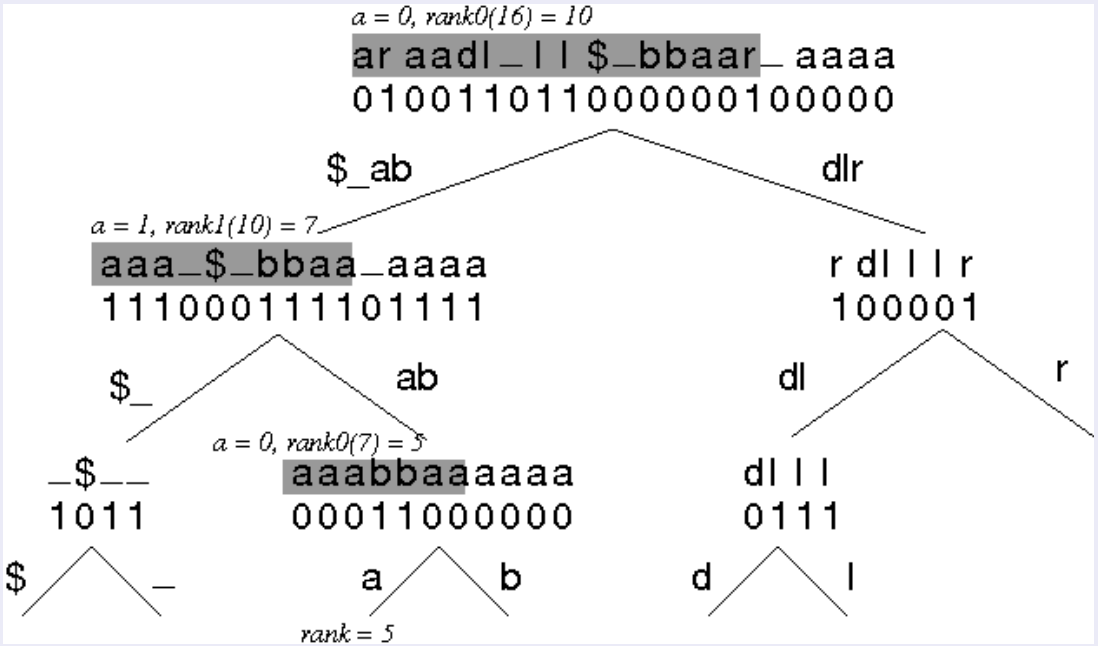
Results

For binary sequences Raman et al. achieves $nH_0(S) + o(n)$ bits and $O(1)$ time per operation.

For general sequences Grossi et al. achieves $n \log \sigma + o(n) \log \sigma$ bits and $O(\log \sigma)$ time per operation. Golynski et al. achieves $n \log \sigma + no(\log \sigma)$ bits and:
(i) $rnk(x) = acc(x) = \log \log x$ and $sel(x) = 1$.
(ii) $rnk(x) = \log \log x \log \log \log x$, $acc(x) = 1$ and $sel(x) = \log \log x$.

Wavelet Trees

Example



Range Queries

Navarro et al. showed how to answer range queries when the wavelet tree represents a permutation, in this work we use virtually the same approach for answering range queries in our data structures.

Outline

Notation

- $\mathcal{R} \subseteq A \times B$: binary relation
- $A = \{1, 2, \ldots, n_1\}$, $B = \{1, 2, \ldots, n_2\}$
- $\mathcal{L} : A \times B \rightarrow L \cup \{\perp\}$
- $L = \{1, 2, \ldots, \ell\}$

Queries

- $\mathcal{L}(a, b)$
- $A(b) = \{a,\ (a, b) \in \mathcal{R}\}$
- $B(a) = \{b,\ (a, b) \in \mathcal{R}\}$
- $R(a_1, a_2, b_1, b_2) = \{(a, b) \in \mathcal{R},\ a_1 \leq a \leq a_2,\ b_1 \leq b \leq b_2\}$
- $\mathcal{L}(j) = \{(a, b) \in \mathcal{R},\ \mathcal{L}(a, b) = j\}$
- The sizes of the sets: $|A(b)|, |B(a)|$, $|R(a_1, a_2, b_1, b_2)|$, and $|\mathcal{L}(j)|$

Representation Idea

Based on the proposal of Barbay et al.

A \ B	1	2	3
1	*1*	*2*	
2		*2*	*2*
3		*1*	

S_B	1	2		2	3		2	
$S_{\mathcal{L}}$	*1*	*2*		*2*	*2*		*1*	
X_B	0	0	1	0	0	1	0	1
X_A	0	1	0	0	0	1	0	1

Result

Theorem

$\mathcal{R}$ can be indexed using $(r + o(r))(\log n_2 + \log \ell + o(\log \ell) + O(1)) + o(n_1 + n_2)$ bits of space, where $r = |\mathcal{R}|$. Queries can be answered in the times shown below, where k is the size of the output. One can choose:
(i) $rnk(x) = acc(x) = \log\log x$ and $sel(x) = 1$.
(ii) $rnk(x) = \log\log x \log\log\log x$, $acc(x) = 1$ and $sel(x) = \log\log x$.

Operation	Time (with range)
$\mathcal{L}(a, b)$	$O(\log n_2 + acc(\ell))$
$A(b)$	$O(1 + k \log n_2)$
$B(a)$	$O(1 + k \log n_2)$
$\lvert A(b)\rvert, \lvert B(a)\rvert$	$O(1)$
$R(a_1, a_2, b_1, b_2)$	$O((k + 1) \log n_2)$
$\lvert R(a_1, a_2, b_1, b_2)\rvert$	$O(\log n_2)$
$\mathcal{L}(l)$	$O((k + 1)sel(\ell) + k \log n_2)$
$\lvert\mathcal{L}(l)\rvert$	$O(rnk(\ell))$

Outline

Powerful SLP Representation

Preconditions

- No repeated rules
- If $X_j \rightarrow \alpha_1$, $X_i \rightarrow \alpha_2$ and $\alpha_1 < \alpha_2$, then $j < i$
- All the symbols in Σ are used

A plain representation requires at least
$2(n-\sigma)\lceil \log n \rceil + \sigma \lceil \log \sigma \rceil \leq 2n \lceil \log n \rceil$ bits

Powerful SLP Representation

Our Representation

Using our binary relation representation, every row represents a symbol X_ℓ and every column a symbol X_r. Pairs (ℓ, r) are related, with label i, whenever there exists a rule $X_i \rightarrow X_\ell X_r$.
Create a bitmap $Y[1, n]$ so that $Y[i] = 1$ if and only if $X_i \rightarrow \alpha$ for some $\alpha \in \Sigma$

Space Usage

The space required by this representation is $2n \log n + o(n \log n)$ bits

Example

Y	ℓ \ r	a	b	d	l	r	A	B	C	D	E	F	G	H	J
1	a				A	C									
1	b								D						
1	d														
1	l														
1	r														
0	A	B													
0	B									E					
0	C														
0	D														
0	E							F							
0	F										G				
0	G			H											
0	H	J													
0	J														

Powerful SLP Representation

Operations

- Access to rules
- Reverse access to rules
- Rules using a left/right symbol
- Rules using a range of symbols

If range is not required, the times can be improved to $O(\log \log n)$

Results

Theorem

An SLP $\mathcal{G} = (X = \{X_1, \ldots, X_n\}, \Sigma)$, $\Sigma = [1, \sigma]$, $\sigma \leq n$, *can be represented using* $2n \log n + o(n \log n)$ *bits, such that all the queries described above (access to rules, reverse access to rules, rules using a symbol, and rules using a range of symbols) can be answered in* $O(\log n)$ *time per delivered datum. If we do not support the rules using a range of symbols, times drop to* $O(\log \log n)$. *For arbitrary integer* Σ *one needs additional* $O(n \log \frac{\max(\Sigma)}{n})$ *bits.*

Outline

LSLP

Definition

A *Lexicographical SLP (LSLP)* $\mathcal{G} = (X, \Sigma, s)$ is a grammar with nonterminals $X = \{X_1, X_2, \ldots, X_n\}$, terminals Σ, and two types of rules:
(i) $X_i \rightarrow \alpha$, where $\alpha \in \Sigma$.
(ii) $X_i \rightarrow X_\ell X_r$. such that:

1. The X_is can be renumbered X'_i in order to obtain an SLP.
2. $\mathcal{F}(X_i) \preceq \mathcal{F}(X_{i+1}), 1 \leq i < n$, being $\preceq$ the lexicographical order.
3. There are no duplicate right hands in the rules.
4. X_s is mapped to X'_n, so that $\mathcal{G}$ represents the text $T = \mathcal{F}(X_s)$.

$$
\begin{array}{lcl|}
A & \longrightarrow & al \\
B & \longrightarrow & ala \\
C & \longrightarrow & ar \\
D & \longrightarrow & bar \\
E & \longrightarrow & alabar \\
F & \longrightarrow & alabarala \\
G & \longrightarrow & alabaralaalabar \\
H & \longrightarrow & alabaralaalabard \\
J & \longrightarrow & alabaralaalabarda \\
\end{array}
$$

A	$\longrightarrow$	al	al
B	$\longrightarrow$	ala	ala
C	$\longrightarrow$	ar	alabar
D	$\longrightarrow$	bar	alabarala
E	$\longrightarrow$	alabar	alabaralaalabar
F	$\longrightarrow$	alabarala	alabaralaalabard
G	$\longrightarrow$	alabaralaalabar	alabaralaalabarda
H	$\longrightarrow$	alabaralaalabard	ar
J	$\longrightarrow$	alabaralaalabarda	bar

Representation of an LSLP

The rows will represent X_ℓ as before, but these will be sorted by *reverse* lexicographic order, as if they represented $\mathcal{F}(X_\ell)^{rev}$. The columns will represent X_r, ordered lexicographically by $\mathcal{F}(X_r)$. We will also store a permutation π_R (using Munro et al.'s representation in order to support π^{-1}), which maps reverse to direct lexicographic ordering.
We store the lenghts of the phrases using $n \log u$ bits.

Example

la	al
ala	ala
rabala	alabar
alarabala	alabarala
rabalaalarabala	alabaralaalabar
drabalaalarabala	alabaralaalabard
adrabalaalarabala	alabaralaalabarda
ra	ar
rab	bar

Example

adrabalaalarabala	al
ala	ala
alarabala	alabar
drabalaalarabala	alabarala
la	alabaralaalabar
ra	alabaralaalabard
rab	alabaralaalabarda
rabala	ar
rabalaalarabala	bar

Occurrence Types

- A primary occurrence in $\mathcal{F}(X_i)$, $X_i \rightarrow X_\ell X_r$, is such that it spans a suffix of $\mathcal{F}(X_\ell)$ and a prefix of $\mathcal{F}(X_r)$
- Every symbol containing X_i (primary occurrence) is a secondary occurrence

Using this, we can focus on finding all the primary occurrences (for each partition) and then collect the secondary occurrences by recursively obtaining the symbols containing the pattern, which requires $O((m + h \cdot occ) \log n)$ time.

Example

Searching for *ar-al* $\Rightarrow$ *al* (direct) and *ra* (reverse) – Omitting terminals

	al (A)	ala (B)	alabar (C)	alabarala (D)	alabaralaalabar (E)	alabaralaalabard (F)	alabaralaalabarda (G)	ar (H)	bar (J)
adrabalaalarabala (G)									
ala (B)									C
alarabala (D)									E
drabalaalarabala (F)									
la (A)									
ra (H)									
rab (J)									
rabala (C)		D							
rabalaalarabala (E)									

Example

Searching for *ar-al* $\Rightarrow$ *al* (direct) and *ra* (reverse) – Omitting terminals

	al (A)	ala (B)	alabar (C)	alabarala (D)	alabaralaalabar (E)	alabaralaalabard (F)	alabaralaalabarda (G)	ar (H)	bar (J)
adrabalaalarabala (G)									
ala (B)									C
alarabala (D)									E
drabalaalarabala (F)									
la (A)									
ra (H)									
rab (J)									
rabala (C)		D							
rabalaalarabala(E)									

Example

Searching for *ar-al* $\Rightarrow$ *al* (direct) and *ra* (reverse) – Omitting terminals

	al (A)	ala (B)	alabar (C)	alabarala (D)	alabaralaalabar (E)	alabaralaalabard (F)	alabaralaalabarda (G)	ar (H)	bar (J)
adrabalaalarabala (G)									
ala (B)									C
alarabala (D)									E
drabalaalarabala (F)									
la (A)									
ra (H)									
rab (J)									
rabala (C)		D							
rabalaalarabala(E)									

Example

Searching for *ar-al* $\Rightarrow$ *al* (direct) and *ra* (reverse) – Omitting terminals

	al (A)	ala (B)	alabar (C)	alabarala (D)	alabaralaalabar (E)	alabaralaalabard (F)	alabaralaalabarda (G)	ar (H)	bar (J)
adrabalaalarabala (G)									
ala (B)									C
alarabala (D)									E
drabalaalarabala (F)									
la (A)									
ra (H)									
rab (J)									
rabala (C)		D							
rabalaalarabala(E)									

Searching 1: Binary Search

- For each prefix of the pattern, we perform a binary search over the direct phrases
- We perform the same using the inverted suffix of the pattern over the reverse phrases
- Determining the range takes $O((m+h)\log^2 n)$ time
- Using the ranges determined by the binary search we recover the primary occurrences (recall range queries)

Searching 2: Compact Patricia Trees

- Index $\mathcal{F}(X_i)$s in a Patricia Tree, using Benoit et al. cardinal tree representation
- Do the same for the reverse phrases
- Space required: $2n \log \sigma + O(n)$ bits per tree
- Problems with the skips, they add $4n \log u$ bits per tree
- We can expand the left- and rightmost children to know the skip, searching takes $O(mh \log n)$ time

Searching 2: Compact Patricia Trees

- We can reserve only k bits for each skip, $[1, 2^k - 1]$
- This improves the time to $O((m + h + \frac{mh}{2^k}) \log n)$
- Space required $4kn$ bits
- For example, if $k = \log h$ then we require $4n \log h$ and searching takes $O((m + h) \log n)$

Theorem

Option 1 - Skips using $k = \log h$ bits

Space: $n(\log u + 3\log n + O(\log \sigma + \log h) + o(\log n))$ bits
Extract: $O((l + h)\log n)$ time
Search: $O((m(m + h) + h \cdot occ)\log n)$ time

Option 2 - Patricia Tree

Space: $n(\log u + 3\log n + O(\log \sigma) + o(\log n))$ bits
Search: $O((m^2 + occ)h \log n)$ time

Option 3 - Binary Search

Space: $n(\log u + 3\log n + o(\log n))$ bits
Search: $O((m(m + h)\log n + h \cdot occ)\log n)$ time

Outline

Conclusions and Future Work

Conclusions

- Powerful representation of binary relations and SLPs
- First index for SLPs (previous results resort on sequential search)
- Application: first index based on Re-Pair

Future Work

- Reduce the $n \log u$ term
- Study new approaches for balancing grammars (the $O(\log u)$ term is a huge space penalty in practice)
- Reduce the $O(m^2)$ time in searching, as done for the LZ78-index
- Support other interesting operations
- Secondary memory - I/O efficient
- More complex search operations

The End

Questions?