Suffix tree construction with minimized branching

MASTER'S THESIS

Peter Bašista

Charles University in Prague
Faculty of Mathematics and Physics

Supervisor: RNDr. Tomáš Dvořák, CSc.
Reviewer: Mgr. Rudolf Kadlec

September 3, 2012

Suffix tree

A tree-like data structure for performing fast search-like operations on strings.

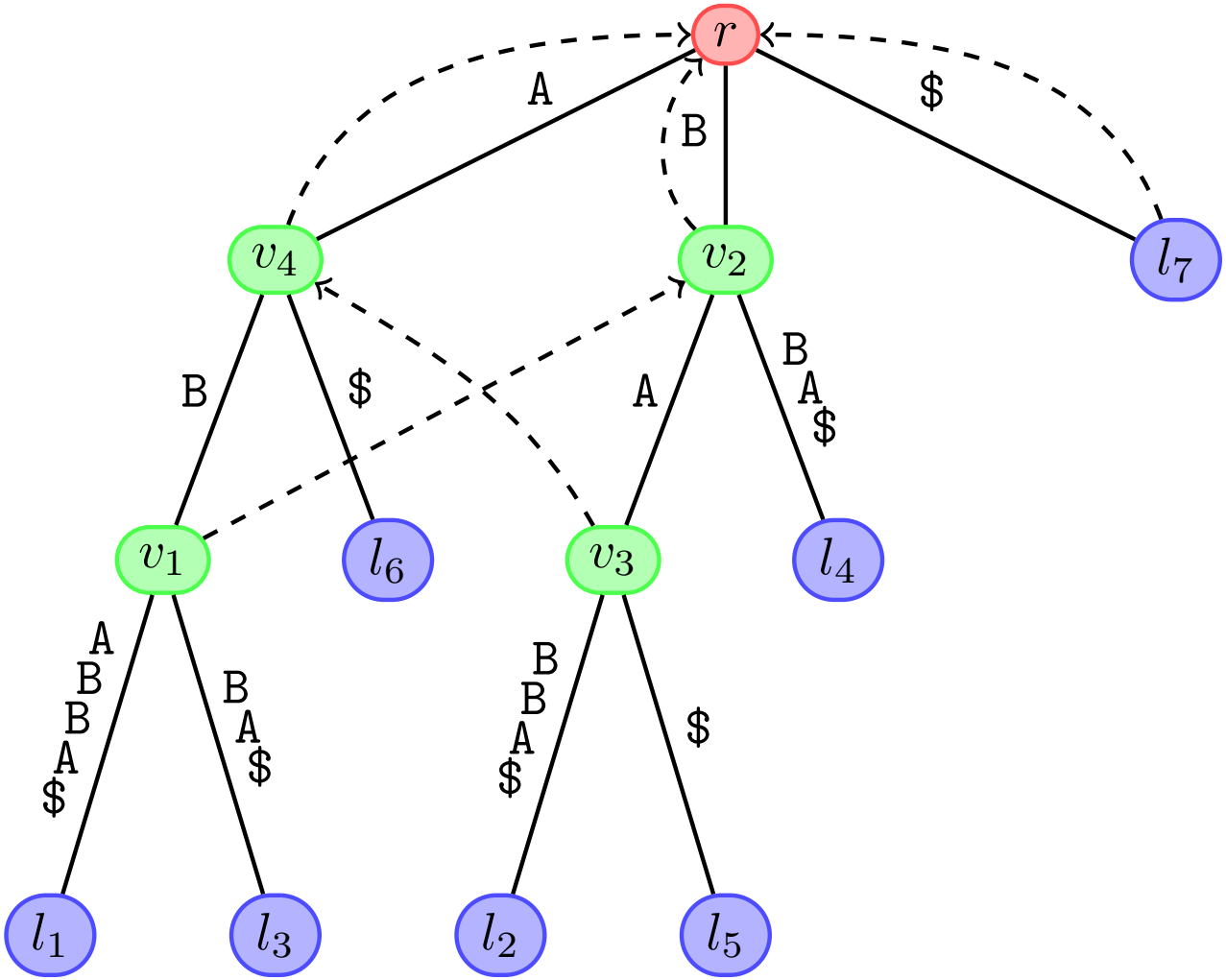
Properties

- Search for a pattern occurrence in $\mathcal{O}(|pattern|)$ time.
- Most implementations require 20–36 bytes per each input character in the worst case.

Some applications

- Bioinformatics
  - searching for patterns in DNA and protein sequences
- Finding repetitive text structures
- Pattern matching using wildcards or regular expressions

Example suffix tree on top of the text ABABBA$



Introduction

Suffix tree construction

- entirely in memory
- over a sliding window

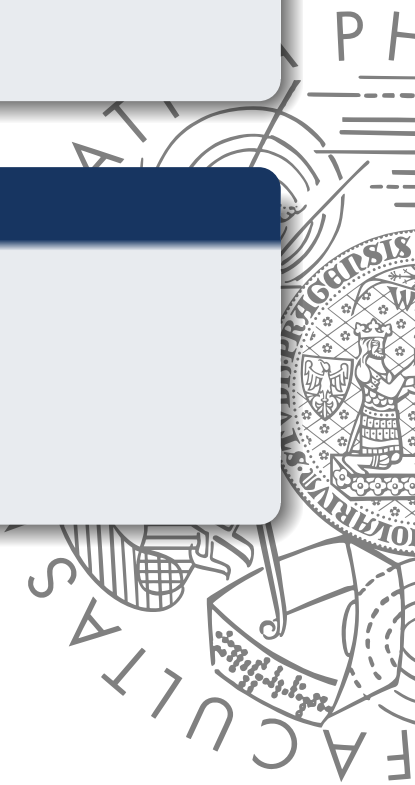
Introduction

Suffix tree construction

- entirely in memory
- over a sliding window

Goals

- theoretical analysis
- implementation
- experimental evaluation



Minimized branching

What is it?

An alternative method for suffix link simulation introduced by Senft and Dvořák, 2012.

Minimized branching

What is it?

An alternative method for suffix link simulation introduced by Senft and Dvořák, 2012.

Suffix link simulation

- Top-down
- *Bottom-up*

Minimized branching

What is it?

An alternative method for suffix link simulation introduced by Senft and Dvořák, 2012.

Suffix link simulation

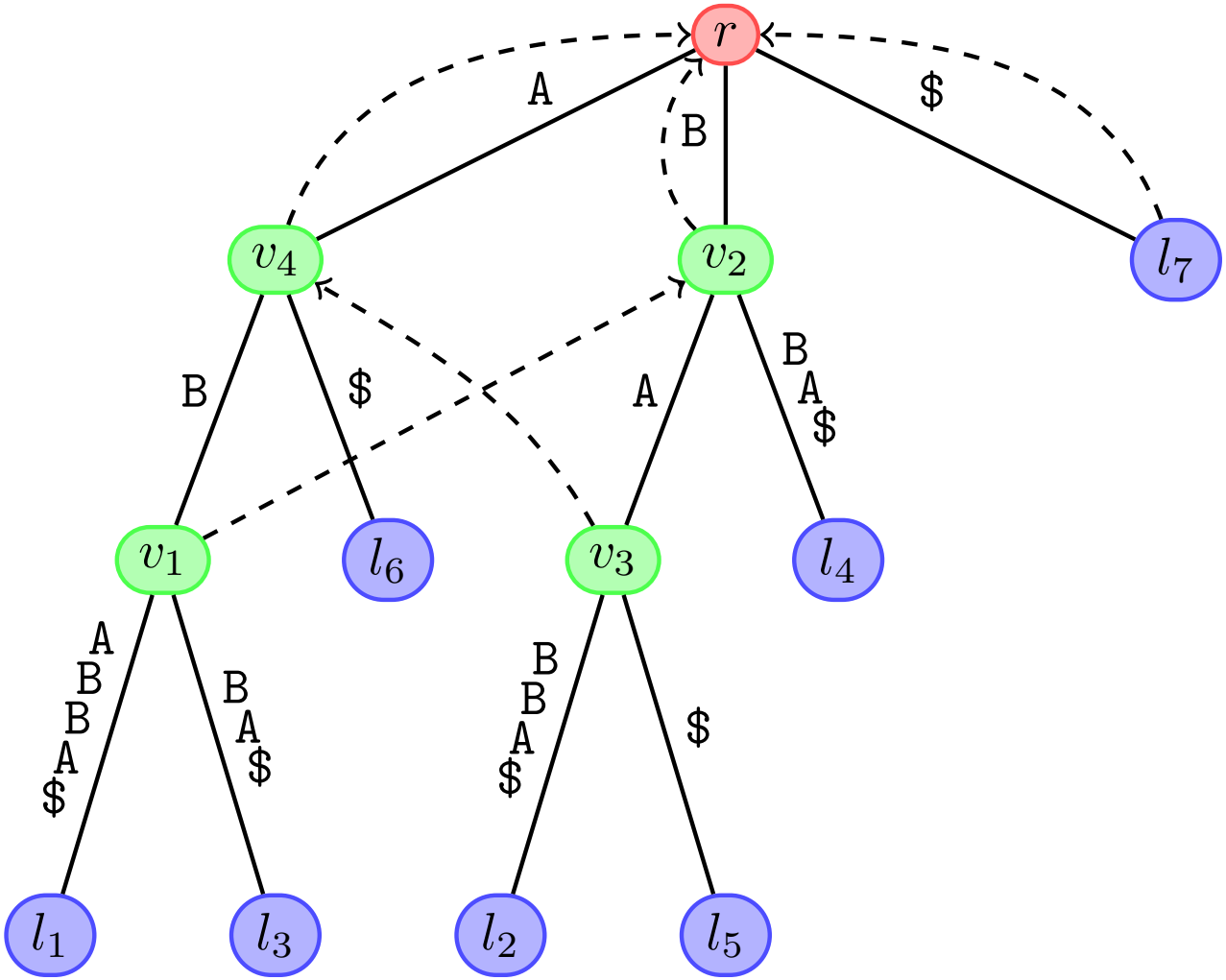
- Top-down
- *Bottom-up*

Applicable to the algorithms by

- McCreight (1976)
- Ukkonen (1992)

Charles University in Prague
Faculty of Mathematics and Physics

Example suffix tree on top of the text ABABBA$



Examined methods of suffix tree construction

Algorithms

- McCreight's (1976)
- Ukkonen's (1992)
- Partition and Write Only Top Down — PWOTD (Tata, Hankins, Patel, 2004)

Examined methods of suffix tree construction

Algorithms

- McCreight's (1976)
- Ukkonen's (1992)
- Partition and Write Only Top Down — PWOTD (Tata, Hankins, Patel, 2004)

Implementation techniques

- Simple Linked List (Kurtz, 1999)
- Simple Hash Table (Kurtz, 1999)
- Simple Linear Array (Tata, Hankins, Patel, 2004)

Experiments conducted

Implementations used

- implementation by Senft and Dvořák, 2012
- SLLI implementation by S. Kurtz, 1999
- PWOTD implementation by Tata, Hankins, Patel, 2004
- our own implementation

Experiments conducted

Implementations used

- implementation by Senft and Dvořák, 2012
- SLLI implementation by S. Kurtz, 1999
- PWOTD implementation by Tata, Hankins, Patel, 2004
- our own implementation

Data used

- pseudorandom input files
- standard corpus files (Pizza & Chili and Lightweight corpus)
- special input files

Results

Description of the algorithms

- We have presented unified descriptions and definitions of every algorithm and its variation analyzed in this thesis.

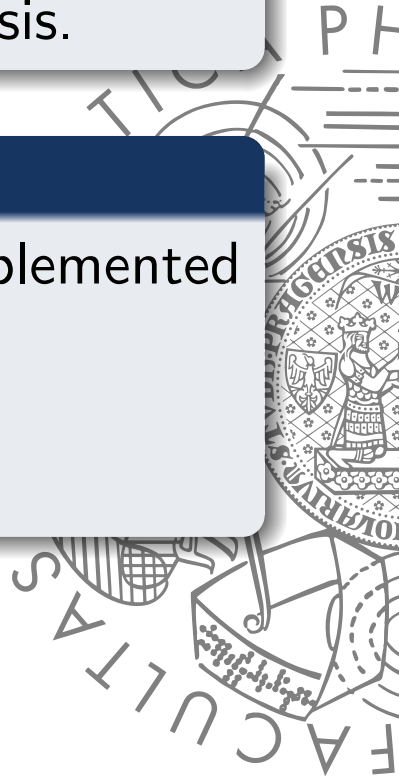
Results

Description of the algorithms

- We have presented unified descriptions and definitions of every algorithm and its variation analyzed in this thesis.

The implementation

- Every algorithm and implementation technique is implemented using similar level of detail and quality.
- Compilable on typical UNIX platforms.
  - tested on Linux and Mac OS X



Results

Usage recommendations

- McCreight's / Ukkonen's algorithms vs. PWOTD
  - PWOTD often computes the length of the lcp
  - therefore it is not suitable for texts with high average lcp

Results

Usage recommendations

- McCreight's / Ukkonen's algorithms vs. PWOTD
  - PWOTD often computes the length of the lcp
  - therefore it is not suitable for texts with high average lcp
- linked lists vs. hash table
  - lower alphabet size $\Longrightarrow$ use linked lists
  - higher alphabet size $\Longrightarrow$ use hash table

Results

Usage recommendations

- McCreight's / Ukkonen's algorithms vs. PWOTD
  - PWOTD often computes the length of the lcp
  - therefore it is not suitable for texts with high average lcp
- linked lists vs. hash table
  - lower alphabet size $\Longrightarrow$ use linked lists
  - higher alphabet size $\Longrightarrow$ use hash table
- percolating update vs. batch update
  - both methods have constant amortized time complexity
  - percolating update makes use of suffix tree traversal during the construction $\Longrightarrow$ its constant is smaller

Experimental results

McCreight's / Ukkonen's algorithms vs. PWOTD

- PWOTD is faster on pseudorandom files
- McCreight's / Ukkonen's algorithms are faster on corpus files
  - PWOTD is slow on files with large average lcp

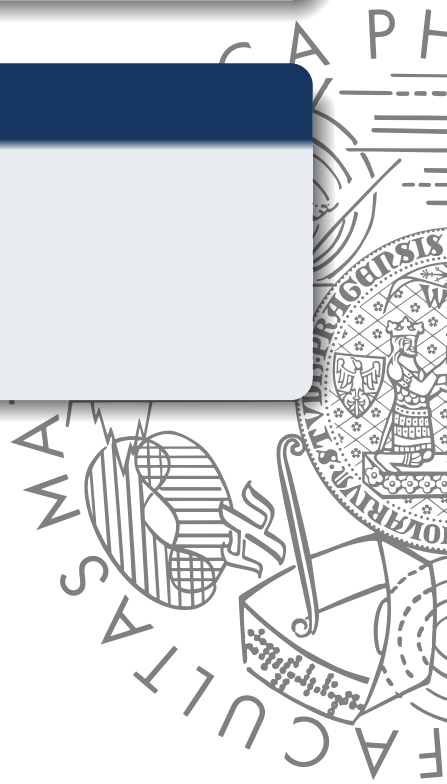
Experimental results

McCreight's / Ukkonen's algorithms vs. PWOTD

- PWOTD is faster on pseudorandom files
- McCreight's / Ukkonen's algorithms are faster on corpus files
  - PWOTD is slow on files with large average lcp

top-down vs. bottom-up suffix link simulation

- bottom-up method *usually* outperforms top-down
- suffix tree construction time is reduced by 5–10%
  - except for special, adversary strings



Experimental results

McCreight's / Ukkonen's algorithms vs. PWOTD

- PWOTD is faster on pseudorandom files
- McCreight's / Ukkonen's algorithms are faster on corpus files
  - PWOTD is slow on files with large average lcp

top-down vs. bottom-up suffix link simulation

- bottom-up method *usually* outperforms top-down
- suffix tree construction time is reduced by 5–10%
  - except for special, adversary strings

percolating update vs. batch update

- percolating update is *almost* always faster
  - pseudorandom files: construction time is reduced by 5–10%
  - corpus files: the reduction is larger, typically around 15%

Suffix tree construction with minimized branching

MASTER'S THESIS

Peter Bašista

Charles University in Prague
Faculty of Mathematics and Physics

Supervisor: RNDr. Tomáš Dvořák, CSc.
Reviewer: Mgr. Rudolf Kadlec

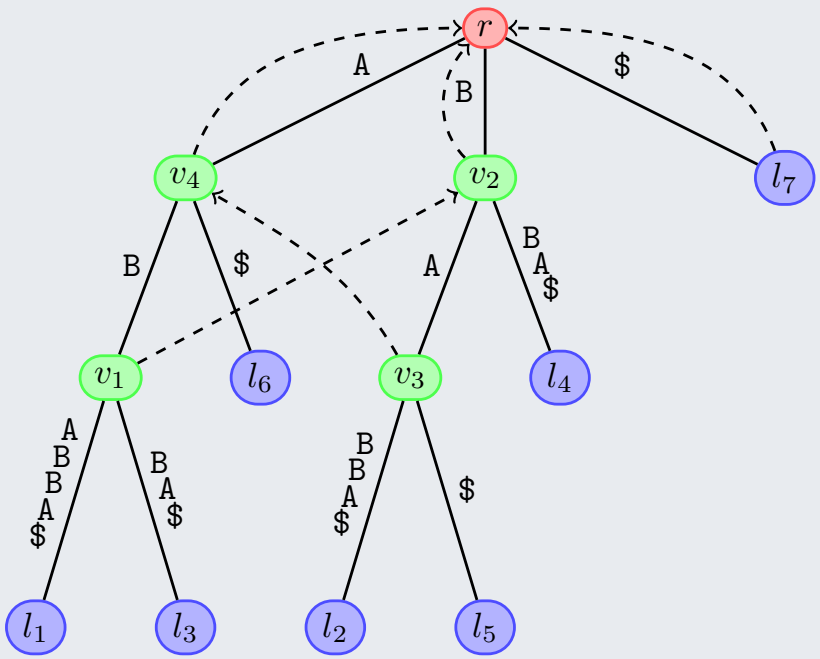
September 3, 2012

Suffix tree traversal log

Human readable suffix tree representation:

`P(number)[depth]--"label"(length)-->C(number)[depth]{suffix link}`

Suffix tree



DFS traversal log

```
Suffix tree traversal BEGIN
P(1)[0]--"a"(1)-->C(5)[1]{1}
P(5)[1]--"b"(1)-->C(2)[2]{3}
P(2)[2]--"abba$"(5)-->C(-1)[7]
P(2)[2]--"ba$"(3)-->C(-3)[5]
P(5)[1]--"$"(1)-->C(-6)[2]
P(1)[0]--"b"(1)-->C(3)[1]{1}
P(3)[1]--"a"(1)-->C(4)[2]{5}
P(4)[2]--"bba$"(4)-->C(-2)[6]
P(4)[2]--"$"(1)-->C(-5)[3]
P(3)[1]--"ba$"(3)-->C(-4)[4]
P(1)[0]--"$"(1)-->C(-7)[1]
Suffix tree traversal END
```

Repetitive structures

maximal pair

a pair of identical substrings impossible to extend in any direction

...GTTATTATTC...

maximal repeat

a substring whose occurrences form a maximal pair

...TACTGACGTTGTC...

supermaximal repeat

A maximal repeat which is not a substring of any other maximal repeat. In the text ACGGCCGTACGA:

- CG is a maximal repeat, but it is not supermaximal
- ACG is a supermaximal repeat