On Compile Time Knuth-Morris-Pratt Precomputation

Justin Kourie[1] Bruce Watson[2,1] Loek Cleophas[3,1]

[1] FASTAR Research Group, Department of Computer Science, University of Pretoria, 0002 Pretoria, Republic of South Africa (justin@fastar.org)

[2] FASTAR Research Group, Centre for Knowledge Dynamics and Decision-making, Stellenbosch University, Private Bag X1, 7602 Matieland, Republic of South Africa (bruce@fastar.org)

[3] Software Engineering & Technology Group, Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands (loek@fastar.org)

Prague Stringology Conference 2011

Outline

1 The General Idea

Outline

1. The General Idea
2. Experiment
   - Benchmark Requirements
   - Implementation Experience

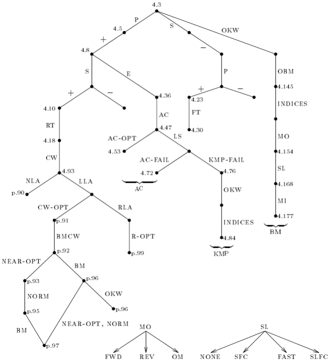
Outline

1. The General Idea
2. Experiment
   - Benchmark Requirements
   - Implementation Experience
3. Analysis
   - Theoretical Speculation
   - Results and Observations

Outline

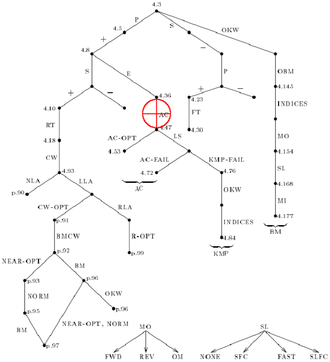
1. The General Idea
2. Experiment
   - Benchmark Requirements
   - Implementation Experience
3. Analysis
   - Theoretical Speculation
   - Results and Observations
4. Final Remarks

Taxonomological Optimization Targets[?]



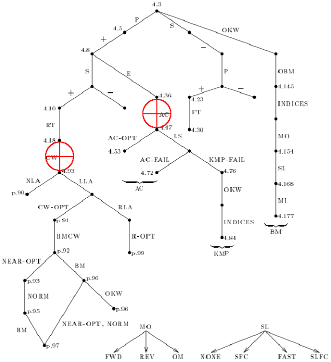
(Watson, BW, *Taxonomies and Toolkits of Regular Language Algorithms*, 1995)

Taxonomological Optimization Targets[?]



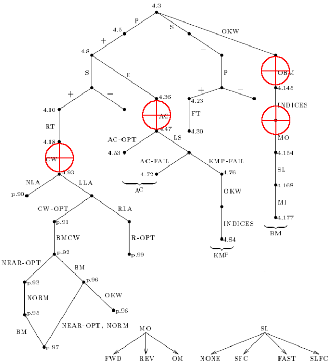
(Watson, BW, *Taxonomies and Toolkits of Regular Language Algorithms*, 1995)

Taxonomological Optimization Targets[?]



(Watson, BW, *Taxonomies and Toolkits of Regular Language Algorithms*, 1995)

Taxonomological Optimization Targets[?]



(Watson, BW, *Taxonomies and Toolkits of Regular Language Algorithms*, 1995)

Keywords Known at Compile Time

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
- *How* can it be implemented?

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
- *How* can it be implemented?

Why KMP?

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
- *How* can it be implemented?

Why KMP?

- Need a starting point to experiment with implementation techniques

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
- *How* can it be implemented?

Why KMP?

- Need a starting point to experiment with implementation techniques
- Start as *simply* as possible

Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
- *How* can it be implemented?

Why KMP?

- Need a starting point to experiment with implementation techniques
- Start as *simply* as possible
- Primary aim was probatory research

Design Overview

Design Overview

Compile time requirements

- Generate a set of keywords (K)

Design Overview

Compile time requirements

- Generate a set of keywords (K)
- For each keyword $k \in K$, precompute KMP fail index

Design Overview

Compile time requirements

- Generate a set of keywords (K)
- For each keyword $k \in K$, precompute KMP fail index

Run time requirements

Design Overview

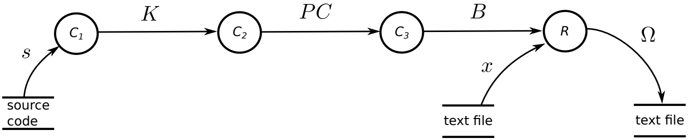
Compile time requirements

- Generate a set of keywords (K)
- For each keyword $k \in K$, precompute KMP fail index

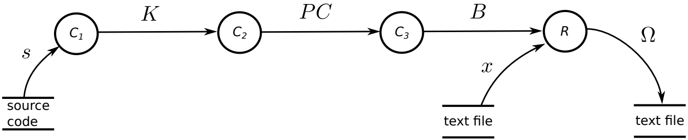
Run time requirements

- Benchmark the following using some target text (x)
  - Traditional (non optimised) KMP
  - Optimised KMP
  - Precomputation algortihm at runtime

A Pipelined Approach

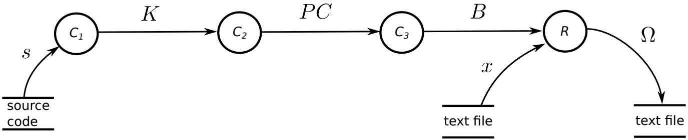


A Pipelined Approach



NOTE: Complexity Analysis VS Complex Experiment

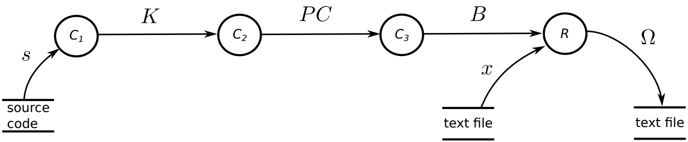
A Pipelined Approach



NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized $= O(n)$, traditional $= O(n+m)$)

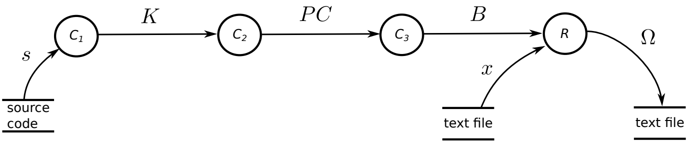
A Pipelined Approach



NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized $= O(n)$, traditional $= O(n+m)$)
- Generated K such that no $k \in K$ present in x

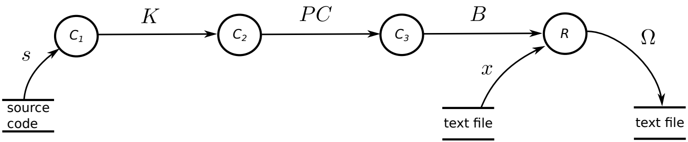
A Pipelined Approach



NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized = $O(n)$, traditional = $O(n+m)$)
- Generated K such that no $k \in K$ present in x
- *Simple* way to build large, consistent data to analyse

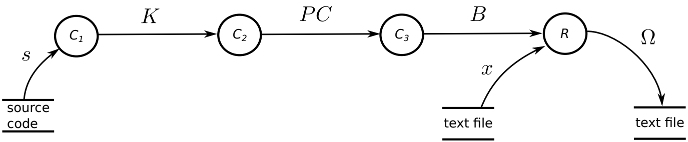
A Pipelined Approach



NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized $= O(n)$, traditional $= O(n + m)$)
- Generated K such that no $k \in K$ present in x
- *Simple* way to build large, consistent data to analyse
- Allowed focus to be on implementation and analysis (not design paralysis)

A Pipelined Approach



NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized $= O(n)$, traditional $= O(n + m)$)
- Generated K such that no $k \in K$ present in x
- *Simple* way to build large, consistent data to analyse
- Allowed focus to be on implementation and analysis (not design paralysis)
- Not ideal, can be improved

C++

Eventualy Hit a Dead End

Severe Metaprogramming Constraints

C++
Eventualy Hit a Dead End

Severe Metaprogramming Constraints

- Constrained string *length*

C++
Eventualy Hit a Dead End

Severe Metaprogramming Constraints

- Constrained string *length*
- Very high computational overhead

C++
Eventualy Hit a Dead End

Severe Metaprogramming Constraints

- Constrained string *length*
- Very high computational overhead
- 'Poor',' wri',tabi',lity'

```
typedef mpl::string<'hell','o wo','rld'> hello;
```

C++

Eventualy Hit a Dead End

Severe Metaprogramming Constraints

- Constrained string *length*
- Very high computational overhead
- 'Poor',' wri',tabi',lity'

```
typedef mpl::string<'hell','o wo','rld'> hello;
```

- Variadic compile time array initialisation

```
fail_idx[] = { precomp<k>::compute };
// i.e., int foo = { 0, 0, 1, ...  }
```

D

Designed for Metaprogramming

Addresses all problems encountered in C++ (length, computation, writability, array initialisation).

D
Designed for Metaprogramming

Addresses all problems encountered in C++ (length, computation, writability, array initialisation).

Why?

D

Designed for Metaprogramming

Addresses all problems encountered in C++ (length, computation, writability, array initialisation).

Why?

- Ground up design = more powerful metaprogramming constructs

D
Designed for Metaprogramming

Addresses all problems encountered in C++ (length, computation, writability, array initialisation).

Why?

- Ground up design = more powerful metaprogramming constructs
- Compile Time Function Evaluation

Hypotheses

In depth discussion of:

Hypotheses

In depth discussion of:

- Sanity checks

Hypotheses

In depth discussion of:

- Sanity checks
- Postulations about usefulness

Hypotheses

In depth discussion of:

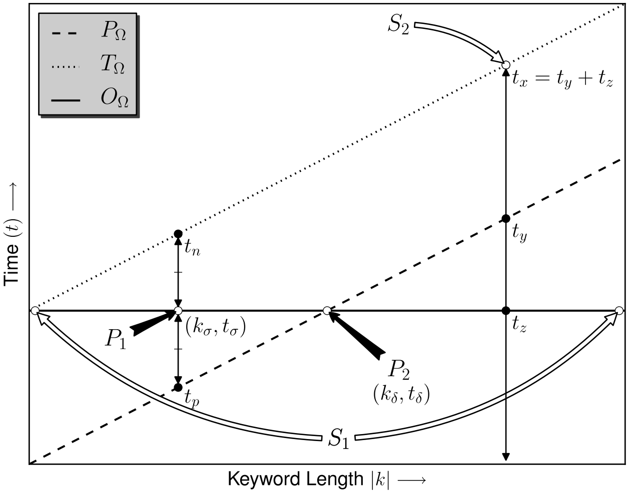
- Sanity checks
- Postulations about usefulness
- No strict claims or generalisations

Hypotheses

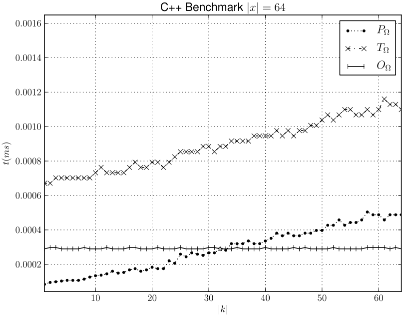
In depth discussion of:

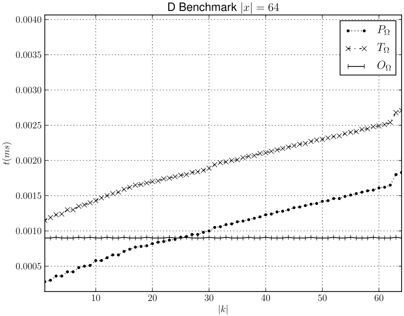
- Sanity checks
- Postulations about usefulness
- No strict claims or generalisations
- Interesting results to observe nonetheless

Hypotheses


P_Ω
T_Ω
O_Ω
S_2
$t_x = t_y + t_z$
t_y
t_n
(k_σ, t_σ)
P_1
t_z
P_2
(k_δ, t_δ)
t_p
S_1
Time (t) ⟶
Keyword Length $|k|$ ⟶

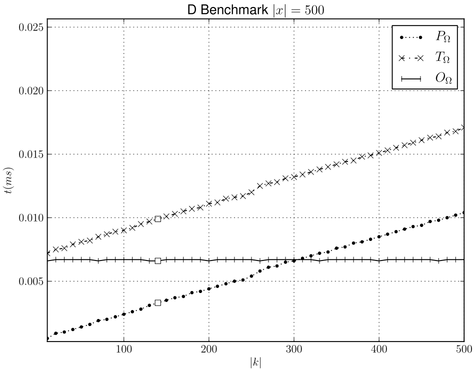
C++ VS D
A Limited Comparison





Interpeting D Data

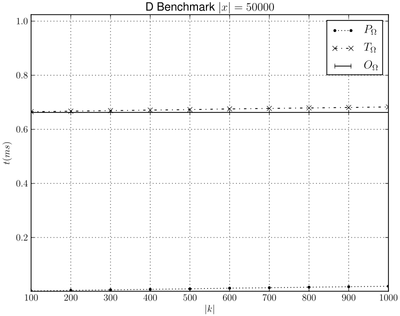
When can we justify Compile Time Optimizations?



- Strong case for optimized search observed where $|k| \geq \approx \frac{|x|}{4}$

Interpeting D Data

When is Compile Time Optimizations Redundant?



- Optimized search gains neared redundancy where $|k| \leq \approx \frac{|x|}{50}$

Answers to the Original Questions

Answers to the Original Questions

When could this technique be useful?

Answers to the Original Questions

When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).

Answers to the Original Questions

When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)

Answers to the Original Questions

When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
- Not strict generalisations nor surprising in themselves but...

Answers to the Original Questions

When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
- Not strict generalisations nor surprising in themselves but...

...How could this technique be implemented?

Answers to the Original Questions

When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
- Not strict generalisations nor surprising in themselves but...

...How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls

Answers to the Original Questions

When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
- Not strict generalisations nor surprising in themselves but...

...How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls
- Choose a language designed for string metaprogramming

Answers to the Original Questions

When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
- Not strict generalisations nor surprising in themselves but...

...How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls
- Choose a language designed for string metaprogramming
- General idea of the benchmark's design

Thanks and Acknowledgements

Thanks and Acknowledgements

- ...and of course questions! $=/$