



# A Taxonomy and Toolkit of Keyword Pattern Matching Algorithms

Loek Cleophas

May 26, 2004

Software Construction Group  
FASTAR Research Group

<http://www.win.tue.nl/fastar>

1. Overview
  2. Problem Introduction
  3. Examples & Categories
  4. Original Taxonomy
  5. New Taxonomy
  6. SPARE Parts: The Original Toolkit
  7. SPARE Time: The New Toolkit
  8. Conclusions
  9. Future Work & Projects
- References



## Solutions & Taxonomies (I)

KPM has many solutions

Watson & Zwaan developed taxonomy and toolkit in early 90s; aims:

- obtain correctness arguments / proofs
- make algorithms easy to understand / compare
- find new algorithms
- create large collection of implementations
- compare running time performance

## Solutions & Taxonomies (II)

Cleophas extended taxonomy and extended & revised toolkit in 2003

- Taxonomy extended;
  - new algorithms described in past decade
  - other algorithms gained attention in past decade
- Toolkit revised & extended
  - C++ language changes: booleans, templates
  - stable STL, providing basic data structures, iterators, ...
  - add algorithms from new taxonomy part to toolkit

## Some Necessary ‘Definitions’

- *prefixes* of a string are strings that appear as substrings at its beginning
- *suffixes* of a string are strings that appear as substrings at its end
- *factors* of a string are strings that appear as substrings

**Note:** This includes the string itself and the empty word  $\varepsilon$ .

**Example:** For the word *baby*,

- the prefixes are baby, bab, ba, b,  $\varepsilon$
- the suffixes are baby, aby, by, y,  $\varepsilon$
- the factors are baby, bab, aby, ba, ab, by, b (occurs as a factor twice), a, y,  $\varepsilon$

## Categories of Pattern Matching - Some Observations and Remarks (I)

### *Forward, Naive*

- each character of text read at least once
- uses trie recognizing prefixes of keywords, or no automaton at all

### *Forward, Prefix-based*

- each character of text read exactly once
- uses automaton which is extension of above trie
- some use bit-parallelism

## Categories of Pattern Matching - Some Observations and Remarks (II)

### *Backward, Suffix-based*

- not all characters of text have to be read
- uses reverse suffix automaton / reverse trie: recognizes keywords' reverse suffixes
- shifts of more than 1 position possible

### *Backward, Factor-based*

- uses factor automaton: larger automaton, recognizes keywords' reverse factors
- reads more characters than suffix-based
- larger shifts than suffix-based
- some use bit-parallelism

## Categories of Pattern Matching - Some Observations and Remarks (III)

### *Backward, Factor Oracle-based*

- uses factor oracle: recognizes at least keywords' reverse factors, possibly more
- reads more characters than factor-based
- same shifts as with factor-based
- easier to construct than factor automaton
- smaller than factor automaton

## Categories of Pattern Matching - “Famous” Algorithms

*Forward, Naive*

*Forward, Prefix-based*

Aho-Corasick,

Knuth-Morris-Pratt,

Shift-And

*Backward, Suffix-based*

Commentz-Walter,

Boyer-Moore,

Boyer-Moore-Horspool

*Backward, Factor-based*

Backward DAWG Matching,

Backward Nondeterm. DAWG Matching

*Backward, Factor oracle-based*

Backward Oracle Matching

*(Filtration-based*

Karp-Rabin and others)

## What Is A Taxonomy?

- Classification according to *algorithm & problem details*
- Algorithm details deal with algorithm structure
- Problem details deal with problem changes
- Tree form (actually, *directed acyclic graph*)
- Nodes are algorithms, branches represent details added
- Root is most simple KPM solution: ‘the solution is to take the set of occurrences of the keywords in the text’
- Refinement by adding details (branches)

## Taxonomy Algorithms (I)

{ Algorithm () (*Root Algorithm*) }

$$O := (\bigcup l, v, r : lvr = S \wedge v \in P : \{(l, v, r)\})$$

$$\{ R : O = (\bigcup l, v, r : lvr = S \wedge v \in P : \{(l, v, r)\}) \}$$

{ Algorithm (P) }

$$O := \emptyset;$$

**for**  $(u, r) : ur = S \rightarrow$

$$O := O \cup (\bigcup l, v : lv = u \wedge v \in P : \{(l, v, r)\})$$

**rof** { R }

{ Algorithm (P,S) }

$$O := \emptyset;$$

**for**  $(u, r) : ur = S \rightarrow$

**for**  $(l, v) : lv = u \rightarrow$

**as**  $v \in P \rightarrow O := O \cup \{(l, v, r)\}$  **sa**

**rof**

**rof** { R }

{ Algorithm (P+,S) }

$$u, r := \varepsilon, S;$$

**if**  $\varepsilon \in P \rightarrow O := \{(\varepsilon, \varepsilon, S)\}$  **||**  $\varepsilon \notin P \rightarrow O := \emptyset$  **fi**

**do**  $r \neq \varepsilon \rightarrow$

$u, r := u(r \uparrow 1), r \downarrow 1;$

**for**  $(l, v) : lv = u \rightarrow$

**as**  $v \in P \rightarrow O := O \cup \{(l, v, r)\}$  **sa**

**rof**

**od** { R }

## Taxonomy Algorithms (II)

```

{ Algorithm (P+, S+) }
u, r := ε, S;
if ε ∈ P → O := {(ε, ε, S)} || ε ∉ P → O := ∅ fi
do r ≠ ε →
    u, r := u(r↑1), r↓1;
    l, v := u, ε;
    as ε ∈ P → O := O ∪ {(u, ε, r)} sa
    do l ≠ ε →
        l, v := l↓1, (l↑1)v;
        as v ∈ P → O := O ∪ {(l, v, r)} sa
    od
od{ R }

```

```

{ Algorithm (P+, S+, GS, EGC, SSD) }
u, r := ε, S;
if ε ∈ P → O := {(ε, ε, S)} || ε ∉ P → O := ∅ fi
l, v := ε, ε;
do r ≠ ε →
    u, r := u(r↑k(l, v, r)), r↓k(l, v, r);
    l, v := u, ε;
    q := δR,f(q0, l↑1);
    as ε ∈ P → O := O ∪ {(u, ε, r)} sa
    { invariant: q = δR,f*(q0, ((l↑1)v)R) }
    do l ≠ ε and q ≠ ⊥ →
        l, v := l↓1, (l↑1)v;
        q := δR,f(q, l↑1);
        as v ∈ P → O := O ∪ {(l, v, r)} sa
    od
od{ R }

```

## Taxonomy Changes

- subtree (P+,S+,RT,SSD) replaced by (P+,S+,GS=...,EGC=...,SSD)
  - used to include suffix-based algorithms
  - now also includes factor-, factor oracle-based algorithms
- in subtree (P+,S+,GS=...,EGC=...,SSD), more shift functions added
- subtree (P+,E,AC) refined to (P+,E,SPP,AC)
  - detail (AC) split into (SPP) and (AC)
  - now includes bit-parallel prefix-based algorithms
  - new algorithm: bit-parallel Aho-Corasick

## SPARE PARTS: The Original Toolkit (I)

- follows structure of taxonomy closely
- implemented in C++:
  - was in widespread use
  - supported generic programming
  - efficient
- shallow class hierarchy, prevents virtual function calls
  - empty pattern matcher class PM, with children PMSingle and PMMulti
  - single-keyword PMKMP and PMBM derived from PMSingle
  - multiple-keyword PMAC and PMCW derived from PMMulti

## SPARE PARTS: The Original Toolkit (II)

- many template classes used:
  - PMAC, takes automaton as argument
  - PMBM, takes shifter, skip loop class as arguments
  - PMCW, takes shifter class as argument
  - automaton classes for PMAC
  - shifter classes for PMBM
  - skip loop classes for PMBM
  - shifter classes for PMCW
  - within each such category, interface is same despite not being inherited
- uses own String, Set, Array classes, Iterators, as C++ Standard Template Library had not yet been standardized

## Bringing SPARE PARTS Up-to-date

- original version did not compile with current C++ compilers
- uses C++ Standard Template Library
  - standardized
  - well-known to C++ programmers / potential users of toolkit
- replaced String, Set, Array, Iterators by STL classes and algorithms;
- fixed issues due to changes in C++ since 1995
  - template code had to be changed in some places
  - use of boolean type instead of integer type used as boolean

## SPARE TIME: The New Toolkit (I)

- toolkit reusability can mean two things:
  - reusability of toolkit components
  - reusability meaning extensibility of toolkit

second meaning important to us;  
SPARE PARTS turned out to be very reusable

- added factor- and factor oracle-based algorithms by generalizing PMCW:
  - added template parameter to PMCW, for type of automaton
  - implemented factor oracle

little work due to structure of toolkit, way in which new algorithms were added to taxonomy

## SPARE TIME: The New Toolkit (II)

- added new shift functions
  - NFS (No-Factor Shifter)
  - BMH (Boyer-Moore-Horspool Shifter)
  - Max (Takes maximum of two Shifters)small amount of work
- not yet implemented
  - bit-parallel forward, prefix-based algorithms; completely new code
  - more new automata: (compact) suffix / factor automaton
  - generalization of existing shifters
- tuning required, new version not as fast as expected, probably due to straightforward STL usage

## Conclusions (I)

- Original taxonomy achieved aims, made extending easy.
- Factor- and factor oracle-based pattern matchers variants of generalized (suffix-based) Commentz-Walter algorithm
- Other algorithms (Shift-And, Shift-Or) and shift functions (Boyer-Moore-Horspool, NFS, Max) derived from taxonomy
- Revising and expanding toolkit easy, original toolkit reusable
- Due to taxonomy-basedness of original toolkit, very structured

## Conclusions (II)

- Made using C++ STL instead of own String, Set... classes straightforward.
- Made generalizing Commentz-Walter pattern matcher and adding new shifters easy
- Implement more algorithms, benchmark and compare; gain insight into when to use which algorithm
- As with original taxonomy and toolkit, use of formal techniques in deriving algorithms and creating a taxonomy of algorithms, helped in comparing and implementing

## Future Work & Projects

- Still some well-known algorithms to add
- Generalize suffix-based shifters to work with other automata
- Toolkit tuning & benchmarking
- Development of ‘little language’ to simplify toolkit use
- Extensions to related fields: approximate, regular expression, tree pattern matching, compressed text

Internship or master’s projects: look at <http://www.win.tue.nl/fastar> for ideas and talk to prof. Watson or other SoC group members!

## References

L.G.W.A. Cleophas, B.W. Watson & G. Zwaan, *A new taxonomy of sublinear keyword pattern matching algorithms*, CS-Report 04-07, TU/e, March 2004.

B.W. Watson & L.G.W.A. Cleophas, *SPARE Parts: a C++ toolkit for string pattern recognition*, *Software—Practice & Experience*, 34(7):697-710, June 2004.

L.G.W.A. Cleophas, *Towards SPARE TIME - A New Taxonomy and Toolkit of Keyword Pattern Matching Algorithms*, MSc thesis, TU/e, August 2003.

B.W. Watson & G. Zwaan, *A taxonomy of sublinear multiple keyword pattern matching algorithms*, *Science of Computer Programming* 27(2):85-118, September 1996.

B.W. Watson, *Taxonomies and Toolkits of Regular Language Algorithms*, PhD thesis, Technische Universiteit Eindhoven, 1995.