Multi-Pattern String Matching with Very Large Pattern Sets

Leena Salmela

L. Salmela, J. Tarhio and J. Kytöjoki: Multi-pattern string matching with q-grams.
ACM Journal of Experimental Algorithmics, Volume 11, 2006.

November 1st 2007

Outline

Problem Definition and Motivation

Previous Algorithms

Filtering Approach
    Verification
    Filtering

Experimental Results

Outline

Problem Definition and Motivation

Previous Algorithms

Filtering Approach
    Verification
    Filtering

Experimental Results

Problem Definition

Definition (Multiple pattern matching problem)

Given a pattern set P and a text t, report all occurrences of all the patterns in the text.

- The text t is a string of n characters drawn from the alphabet Σ (of size σ).
- The pattern set P is a set of r patterns each of which is a string of characters over the alphabet Σ.
- For simplicity we assume that all patterns have the same length m.
- We are especially interested in searching for large pattern sets ($>$10,000 patterns)

Why large pattern sets?

- Applications where large pattern sets are needed:
    - Antivirus scanning (around 100,000 known viruses)
    - Intrusion detection
    - Bioinformatics
- Older algorithms were not developed for such large pattern sets and they do not scale very well.

Outline

Trie-Based Algorithms

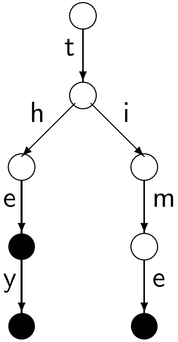


Figure: Trie built of the patterns "the", "they" and "time".

- Aho-Corasick, Commentz-Walter, SBOM etc.
- Many multi pattern algorithms build a trie of the patterns and search the text with the aid of the trie.
- The trie grows quite rapidly as the pattern set grows.
  - For $\sigma = 256$, $m = 8$ and 100,000 patterns the trie takes 500 MB of memory.

$\Longrightarrow$ Trie-based algorithms are not practical for large pattern sets.

Rabin-Karp (for Single Pattern)

Preprocessing

1. Compute a hash of the pattern $hs(p_0 ... p_{m-1})$

Searching

1. For each text position i compute the hash $hs(t_i ... t_{i+m-1})$
2. If the hash equals the hash of the pattern, verify the match.

Multiple Pattern Matching Based on Rabin-Karp

Preprocessing

1. Compute the hash of each pattern $hs(p^i_0 \ldots p^i_{m-1})$ and store them.
2. Sort the patterns according to the hash values.

Searching

1. For each text position i compute the hash $hs(t_i \ldots t_{i+m-1})$
2. Search for the hash value from the saved hash values of the patterns using binary search.
3. If the hash equals the hash of a pattern, verify the match.

Outline

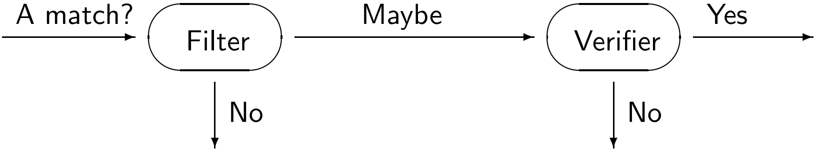
Problem Definition and Motivation

Previous Algorithms

Filtering Approach
- Verification
- Filtering

Experimental Results

Filtering Approach



- Given a text position, a filter can tell if there cannot be a match at this position.
- The hashes in the (single pattern) Rabin-Karp algorithm act as a filter; If the hashes do not match there cannot be a match at that position.
- A good filter is fast and produces few false positives.
- A verifier is needed to distinguish between false and true positives.

Verification

- Verification of a single pattern is easy. (pairwise comparison)
- In a multiple pattern algorithm, the filter only tells some of the patterns might match
  $\Longrightarrow$ The verifier also needs to figure out which pattern to try.
- Using a trie would work but needs a lot of space (something we wanted to avoid in the first place)
- The verifier should be space-efficient and faster than pairwise comparison of all patterns against the given text position
  $\Longrightarrow$ Rabin-Karp for multiple patterns!

Character Class Filter

- Given a set of patterns...

p	a	t	t	e	r	n
f	i	l	t	e	r	s

Character Class Filter

- Given a set of patterns...
- ...construct a generalized pattern with character classes and apply any algorithm capable of handling such generalized patterns.

p a t t e r n
f i l t e r s

↓

[f,p] [a,i] [l,t] [t] [e] [r] [n,s]

Character Class Filter

- Given a set of patterns...
- ...construct a generalized pattern with character classes and apply any algorithm capable of handling such generalized patterns.
- How to make it work with very large pattern sets?

p a t t e r n
f i l t e r s

↓

[f,p] [a,i] [l,t] [t] [e] [r] [n,s]

Character Class Filter with *q*-Grams

- Given a set of patterns...
- ...construct a generalized pattern with character classes and apply any algorithm capable of handling such generalized patterns.
- How to make it work with very large pattern sets?
  - Use superalphabets (q-grams)

p a t t e r n → pa at tt te er rn

f i l t e r s → fi il lt te er rs

Character Class Filter with *q*-Grams

- Given a set of patterns...
- ...construct a generalized pattern with character classes and apply any algorithm capable of handling such generalized patterns.
- How to make it work with very large pattern sets?
  - Use superalphabets (q-grams)
  - ...and construct a generalized pattern.

p a t t e r n → pa at tt te er rn
f i l t e r s → fi il lt te er rs

↓

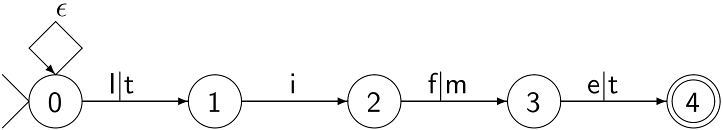
[fi,pa] [at,il] [lt,tt] [te] [er] [rn,rs]

Character Class Filters

- The character class filter is truly a filter
    - It recognizes any occurrence of the pattern.
    - False positives are also found. (I.e. “filtern” and “patters” are recognized by the filter on the previous slide.)
- We have implemented the filter with three different character class algorithms:
    - Multi-Pattern Horspool with q-Grams (HG)
        - A Boyer-Moore-Horspool type algorithm
    - Multi-Pattern Shift-Or with q-Grams (SOG)
        - Shift-Or (simplest, presented in following slides)
    - Multi-Pattern BNDM with q-Grams (BG)
        - BNDM (average optimal for $q = O(\log_\sigma r)$, fastest in practise)

Shift-Or Character Class Filter

- Suppose we are searching for patterns “lift” and “time” so the character class pattern is “[l,t][i][f,m][e,t]”.
- The following NFA finds all occurrences of the character class pattern:



- The shift-or algorithm is a bit-parallel simulation of this automaton.

Shift-Or Character Class Filter: Preprocessing

- For each character c of the alphabet, initialize a bit vector $T[c]$ such that the i'th bit is 0 iff the character appears in any of the patterns in position i.
- In our example (patterns "lift" and "time"):

T['e']	0111
T['f']	1011
T['i']	1101
T['l']	1110
T['m']	1011
T['t']	0110

- The automaton has a transition from state i to state $i + 1$ on character c iff i:th bit in $T[c]$ is 0.

Shift-Or Character Class Filter: Searching

- State vector E where i'th bit is 0 iff state i in the automaton is active.
- Initialize E as 1111.
- Update E when a character c is read from the text:

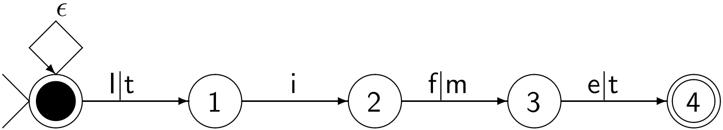
$$E = (E \ll 1) \mid T[c]$$

- After the update, i'th bit in E is 0 iff $i-1$:th bit was 0 (the previous state $i-1$ was active) and i'th bit is 0 in $T[c]$ (there is a transition from state $i-1$ to i on c).

Shift-Or Character Class Filter: Searching

Matching against the text: “ttime”

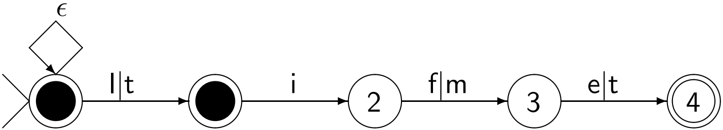
E = 1111



Shift-Or Character Class Filter: Searching

Matching against the text: “ttime”

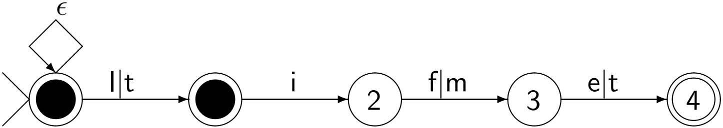
$$\begin{array}{ll} & E = 1111 \\ \text{Read 't'} & E = (1111 \ll 1) \mid 0110 = 1110 \end{array}$$



Shift-Or Character Class Filter: Searching

Matching against the text: “ttime”

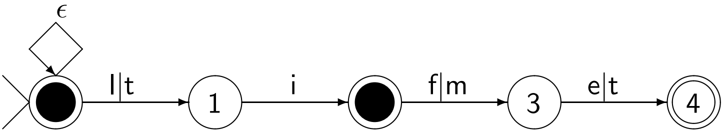
	E = 1111
Read ’t’	E = (1111 ≪ 1) | 0110 = 1110
Read ’t’	E = (1110 ≪ 1) | 0110 = 1110



Shift-Or Character Class Filter: Searching

Matching against the text: “ttime”

$$\begin{array}{ll} & E = 1111 \\ \text{Read 't'} & E = (1111 \ll 1) \mid 0110 = 1110 \\ \text{Read 't'} & E = (1110 \ll 1) \mid 0110 = 1110 \\ \text{Read 'i'} & E = (1110 \ll 1) \mid 1101 = 1101 \end{array}$$



Shift-Or Character Class Filter: Searching

Matching against the text: “ttime”

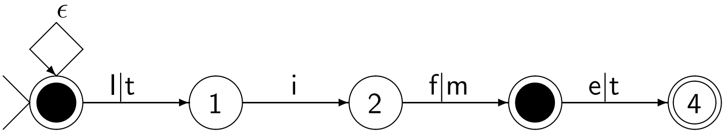
$E = 1111$

Read 't' $E = (1111 \ll 1) \mid 0110 = 1110$

Read 't' $E = (1110 \ll 1) \mid 0110 = 1110$

Read 'i' $E = (1110 \ll 1) \mid 1101 = 1101$

Read 'm' $E = (1101 \ll 1) \mid 1011 = 1011$



Shift-Or Character Class Filter: Searching

Matching against the text: “ttime”

$E = 1111$

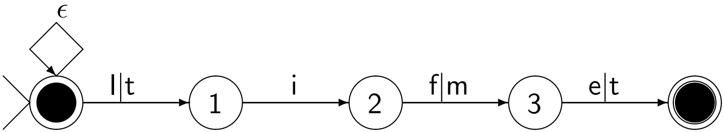
Read 't' $E = (1111 \ll 1) \mid 0110 = 1110$

Read 't' $E = (1110 \ll 1) \mid 0110 = 1110$

Read 'i' $E = (1110 \ll 1) \mid 1101 = 1101$

Read 'm' $E = (1101 \ll 1) \mid 1011 = 1011$

Read 'e' $E = (1011 \ll 1) \mid 0111 = 0111$



Outline

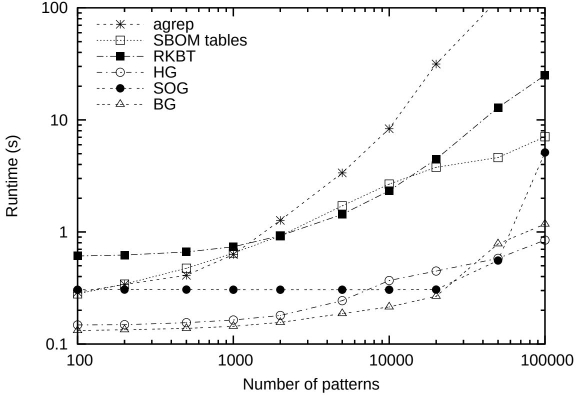
Problem Definition and Motivation

Previous Algorithms

Filtering Approach
- Verification
- Filtering

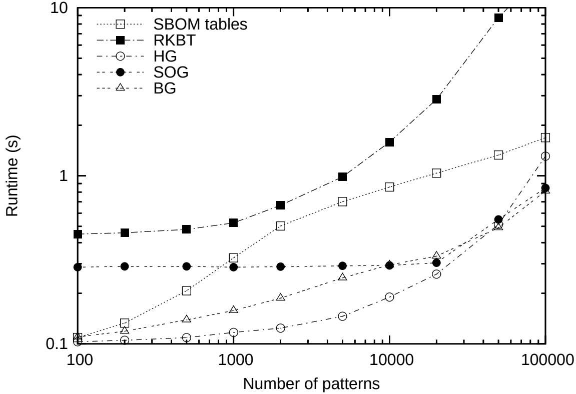
Experimental Results

Experimental Results



$m = 8$, $\sigma = 256$, random data, $q = 2...3$

Experimental Results



$m = 32$, $\sigma = 4$, DNA data (chromosome from fruitfly genome), $q = 6 \ldots 10$

Summary

- Trie-based approaches not practical with very large pattern sets
- Filtering approach to multiple pattern matching
  - Transform patterns to sequences of q-grams
  - Filter with a character class pattern built from the transformed pattern set
  - Verify with a Rabin-Karp style algorithm