Tuning BNDM with $q$-Grams

Branislav Ďurian, Jan Holub, Hannu Peltola, and Jorma Tarhio

Branislav.Durian@snt.sk, holub@fel.cvut.cz, {hpeltola, tarhio}@cs.hut.fi

S&T Varias s.r.o., Priemyselná 2, SK-010 01 Žilina, Slovakia
Department of Computer Science and Engineering,
Czech Technical University in Prague, Karlovo nám. 13, CZ-121 35,
Prague 2, Czech Republic
Department of Computer Science and Engineering, Helsinki University
of Technology, P.O. Box 5400, FI-02015 HUT, Finland
Overview

- Problem
- $q$-grams
- BNDM
- Our variations
- Example of performance
- Tips for speedup
Problem

- Text of length $n$: $T = t_1 t_2 \cdots t_n$
- Pattern of length $m$: $P = p_1 p_2 \cdots p_m$
- Exact pattern matching on strings: find all positions where the given pattern can be found in the text
- Text: English, genome, protein sequence, ...
- What is the right algorithm for the given task given $m$, $n$, alphabet (size and distribution)?
$q$-grams

- Strings of $q$ characters
- 3-grams in ALENEX09 are ALE, LEN, ENE, NEX, EX0, and X09
  (3-gram in LSD is LSD, 3-gram in LAW is LAW)
Pattern matching in strings
– general idea

place pattern in the beginning of the text
Pattern matching in strings
– general idea

place pattern in the beginning of the text while pattern is not past the end of the text
Pattern matching in strings
– general idea

place pattern in the beginning of the text
while pattern is not past the end of the text
check whether here is an occurrence
Pattern matching in strings
– general idea

place pattern in the beginning of the text
while pattern is not past the end of the text
check whether here is an occurrence
shift pattern forward
Pattern matching in strings
– general idea

place pattern in the beginning of the text
while pattern is not past the end of the text
check whether here is an occurrence
shift pattern forward
Pattern matching in strings
– general idea

place pattern in the beginning of the text
while pattern is not past the end of the text
check whether there is an occurrence
shift pattern forward
Bitparallel algorithms

• Simulating an automata in *bitvectors* with various bit operations: shifting, complementing, masking, AND, OR, . . .

• Shift-Or — Shift-And

• Efficient as long as bitvectors fit in one register and memory bus width

• BNDM – Backward Nondeterministic DAWG Matching
\[ P = abac, \ T = abaaabacaabac \]
\[ BNDM(P = p_1 p_2 \cdots p_m, T = t_1 t_2 \cdots t_n) \]

1: \textbf{for all} \( a \in \Sigma \) \textbf{do} \( B[a] \leftarrow 0 \)
2: \textbf{for} \( j \leftarrow 1 \) \textbf{to} \( m \) \textbf{do}
3: \hspace{1em} \( B[p_j] \leftarrow B[p_j] \; | \; (1 \ll (m - j)) \)
4: \( i \leftarrow 0 \)
5: \textbf{while} \( i \leq n - m \) \textbf{do}
6: \hspace{1em} \( j \leftarrow m; \; \text{last} \leftarrow m; \; D \leftarrow (1 \ll m) - 1 \)
7: \hspace{1em} \textbf{while} \( D \neq 0 \) \textbf{do}
8: \hspace{2em} \( D \leftarrow D \& B[t_{i+j}]; \; j \leftarrow j - 1 \)
9: \hspace{2em} \textbf{if} \( D \& (1 \ll (m - 1)) \neq 0 \) \textbf{then}
10: \hspace{3em} \textbf{if} \( j > 0 \) \textbf{then}
11: \hspace{4em} \( \text{last} \leftarrow j \)
12: \hspace{3em} \textbf{else}
13: \hspace{4em} \textbf{report occurrence at} \( i + 1 \)
14: \hspace{2em} \( D \leftarrow D \ll 1 \)
15: \hspace{1em} \( i \leftarrow i + \text{last} \)
Generalization of BNDM

Idea: first $q - 1$ tests of state vector $D$ are skipped

After shift, start with an unrolled evaluation of $q$-gram:

$$D \leftarrow B[t_i] \& (B[t_{i+1}] \ll 1) \& \cdots \& (B[t_{i+q-1}] \ll (q - 1))$$

Shift $\leq m - q + 1$, $q \leq m$

Best values of $q$: 2 – 6

Good for English and DNA
$SBNDM_q$

- Simplified BNDM: records factors instead of prefixes
- We need only to watch that state vector $D \neq 0$, because it comes zero after shifting $m$ times
- Small speedup on small values of $q$
versions of $\text{BNDM}_q$ and $\text{SBNDM}_q$

- A 16-bit halfword (instead of two characters) is read and used in indexing
- Saves two indexed memory reads, shifting, and AND-operation
- Speed penalty up to 70% on reads from memory while crossing 32-bit border
- Initialization takes more effort
- Good on long texts
FNDM

TNDM in [PT2003]

- forward and backward matching
- less accesses than BNDM
- running time worse than BNDM (too complicated code)

FNDM (Forward Nondeterministic DAWG Matching)

- Forward scanning for finding suffixes dominates.
- When we find a suffix, naive search begins.
- Thus we get compact and efficient code.
$P = abac$, $T = abaaabacaabac$
UFNDM_q

- FNDM – Forward Nondeterministic DAWG Matching
- Shifts at least \( q \) positions and when needed also verifies \( q \) promising alignments
- Remembers after shift potential alignments
- \( m + q \leq w \) (computer word length)
- Best on binaries with values of \( q \) up to 10 and good also on DNA patterns
Results for DNA sequences

Search times for 200 DNA patterns

Pattern length (nucleotides)

Time (milliseconds)

BNDMnr
SBNDM2x
SBNDM2b
Lec3
KS
UFNDM5
SBNDM4b
SBNDM6b
Search times for 200 DNA patterns

Pattern length (nucleotides)

Time (milliseconds)

BNDMnr
Lec3
KS
UFNDM5
SBNDM4b
SBNDM6b

Zoomed in results for long DNA sequences
Bitparallel competitors

- Shift-Or: reads every text character; good with short patterns on small alphabets
- SBNDM2x: early version in Lecroq’s tests
- BLIM – Bit-parallel Length Invariant Matcher / Külekci: designed for long patterns
- WW_LBNDM – “Wide Window”; checks $m$ alignments simultaneously and then shifts $m$ positions
- FAOSO – Fast Average Optimal Shift-Or / Fredriksson; two parameters to be adjusted
Other competitors

- Lec$n$: Lecroq’s “New”; hashing uses modulo with power of 2; Fast when hashing in byte
- KS / Kim–Shawe-Taylor: uses a trie of reversed 5-grams of the pattern; Good on DNA (on modern computers)
Normal efficiency indicators may be misleading

- Number of fetched characters explains only little: e.g. in all tests BNDM3 fetches more characters than BNDM2, but is still clearly faster
- Also in all tests the average shift length of BNDM3 is smaller than on BNDM2
- Average shift length differs less than 25%, when $m \geq 20$
Concluding remarks

• Here the use of $q$-grams tries to keep loop running instead of maximizing the average shift length
• Testing of conditions is not free of charge
• String matching results are data dependent
**Tips for speedup**

- Start loop with $q$-grams instead of single character
Tips for speedup

- Start loop with $q$-grams instead of single character
- Choose $q$ based on effective alphabet size and pattern length $m$
Tips for speedup

- Start loop with $q$-grams instead of single character
- Choose $q$ based on effective alphabet size and pattern length $m$
- Read a halfword instead of two separate bytes/characters
**Tips for speedup**

- Start loop with $q$-grams instead of single character
- Choose $q$ based on effective alphabet size and pattern length $m$
- Read a halfword instead of two separate bytes/characters
- Typically compiler takes care about constant offsets in indexes: $B[i-2]$ is equally fast as $B[j]$, but usually $B[i+j]$ is not
Tips for speedup

- Start loop with $q$-grams instead of single character
- Choose $q$ based on effective alphabet size and pattern length $m$
- Read a halfword instead of two separate bytes/characters
- Typically compiler takes care about constant offsets in indexes: $B[i-2]$ is equally fast as $B[j]$, but usually $B[i+j]$ is not
- Straightforward implementation is often fastest
Questions and Advertising

• Any question?

The Prague Stringology Conference 2009
• Prague, August 31–September 2, 2009
• www.stringology.org