Computing Patterns in Strings I: Specific, Generic, Intrinsic

Bill Smyth¹,²,³

¹ Algorithms Research Group, Department of Computing & Software
McMaster University, Hamilton, Ontario, Canada
email: smyth@mcmaster.ca

² Digital Ecosystems & Business Intelligence Institute
Curtin University, Perth, Western Australia
email: b.smyth@curtin.edu.au

³ Department of Computer Science
King’s College London, UK

DEBII 2008
1 Abstract
2 Introduction
   • What is a string?
   • Why are strings important?
   • Examples
   • String conferences
   • Important ideas
3 Computing Specific Patterns
4 Exact Pattern-Matching
   • Skipping — KMP
   • Skipping — Sunday Variant of Boyer-Moore
   • Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
   • Hybrid — FJS & SWY
5 Approximate Pattern-Matching
Abstract

Introduction
- What is a string?
- Why are strings important?
- Examples
- String conferences
- Important ideas

Computing Specific Patterns

Exact Pattern-Matching
- Skipping — KMP
- Skipping — Sunday Variant of Boyer-Moore
- Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
- Hybrid — FJS & SWY

Approximate Pattern-Matching
Outline

1. Abstract
2. Introduction
   - What is a string?
   - Why are strings important?
   - Examples
   - String conferences
   - Important ideas
3. Computing Specific Patterns
4. Exact Pattern-Matching
   - Skipping — KMP
   - Skipping — Sunday Variant of Boyer-Moore
   - Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
   - Hybrid — FJS & SWY
5. Approximate Pattern-Matching

Bill Smyth
Computing Patterns in Strings I
Abstract

Introduction

What is a string?
Why are strings important?
Examples
String conferences
Important ideas

Computing Specific Patterns

Exact Pattern-Matching

Skipping — KMP
Skipping — Sunday Variant of Boyer-Moore
Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
Hybrid — FJS & SWY

Approximate Pattern-Matching
Computing patterns in strings constitutes the combinatorial nuts and bolts of many more general technologies: pattern “recognition”, data mining, data compression, bioinformatics, cryptography, information retrieval, security systems.

In this series of three lectures, I give a nontechnical overview, guaranteed intelligible to the non-mathematician, of these methods, organized into three categories:

* **specific** patterns (pattern-matching);
* **generic** patterns (“regularities” in strings);
* **intrinsic** patterns (always there, they make things happen!).
Outline

1. Abstract
2. Introduction
   - What is a string?
   - Why are strings important?
   - Examples
   - String conferences
   - Important ideas
3. Computing Specific Patterns
4. Exact Pattern-Matching
   - Skipping — KMP
   - Skipping — Sunday Variant of Boyer-Moore
   - Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
   - Hybrid — FJS & SWY
5. Approximate Pattern-Matching
What is a string?

A string is just a sequence of “letters” (symbols) drawn from some (finite or infinite) “alphabet” (set):

- a word in the English language, whose letters are the upper and lower case English letters;
- a text file, whose letters are the ASCII characters;
- a book written in Chinese, whose letters are Chinese ideograms;
- a computer program, whose elements are certain “separators” (space, semicolon, colon, and so on) together with the “words” between separators; also a compiled .exe program;
- a DNA sequence, perhaps three billion letters long, containing only the letters C, G, A and T, standing for the nucleotides cytosine, guanine, adenine and thymine, respectively;
- a stream of trillions of bits beamed from a space vehicle;
- a list of the lengths of the sides of a convex polygon, whose values are drawn from the real numbers.
Why are strings important?

Because **everything** is a string!
Examples

* Fibonacci

\[ f = a \ b \ a \ a \ b \ a \ b \ a \ a \ b \ a \ a \ b \ \ldots \]

* WWW (courtesy Lewis Carroll)

\[ 'Twas \ brillig \ and \ the \ slithy \ toves \ did \ gyre \ and \ gimble \ \ldots \]

* highly periodic

\[ 0010100101101001010011010010010100 \ \ldots \]
Conferences on String Processing

- AFL: International Conference on Automata & Formal Languages
- CIAA: International Conference on Implementation & Application of Automata;
- CPM: Symposium on Combinatorial Pattern Matching;
- DLT: Developments in Language Theory;
- ECCB: European Conference on Computational Biology;
- FSMNLP: Finite-State Methods & Natural Language Processing;
- LATA: International Conference on Language & Automata Theory & Applications;
- LSD: London Stringology Days;
- PSC: Prague Stringology Conference;
- SPIRE: Symposium on String Processing and Information Retrieval;
- StringMasters (@ McMaster): “How long is a piece of string?”
- WABI: Workshop on Algorithms in Bioinformatics;
- WORDS: International Conference on Words.

In 1980 AFL started, the next one was CPM in 1990 — all the others have started since then.
Important Ideas

- combinatorial
- specific
- generic
- intrinsic
Outline

1 Abstract
2 Introduction
   - What is a string?
   - Why are strings important?
   - Examples
   - String conferences
   - Important ideas
3 Computing Specific Patterns
4 Exact Pattern-Matching
   - Skipping — KMP
   - Skipping — Sunday Variant of Boyer-Moore
   - Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
   - Hybrid — FJS & SWY
5 Approximate Pattern-Matching
The Pattern-Matching Task

The problem: to find all occurrences of a given pattern \( p = p[1..m] \) in a given text \( x = x[1..n] \) – hundreds, if not thousands, of algorithms have been proposed. More than 30 are given, with descriptions and C code, at

http://www-igm.univ-mlv.fr/~lecroq/string/index.html

Two approaches:

(1) skip sections of \( x \) where \( p \) cannot occur;
(2) use the bit-parallel property of computer words to go through \( x \) fast!
(3) hybrid – do first one, then the other.

Two modes:

(1) exact – issi occurs twice in Mississippi;
(2) approximate – ipsi occurs three times with one error in Mississippi!
Abstract

Introduction

What is a string?
Why are strings important?
Examples
String conferences
Important ideas

Computing Specific Patterns

Exact Pattern-Matching

Skipping — KMP
Skipping — Sunday Variant of Boyer-Moore
Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
Hybrid — FJS & SWY

Approximate Pattern-Matching
The Knuth-Morris-Pratt Algorithm

- The most famous pattern-matching algorithm.
- Preprocessing: compute the **border** of every prefix of \( p \).
- Requires at most \( 2n \) letter comparisons (linear!).
- However, not very fast in practice.
The KMP Algorithm - 1
The KMP Algorithm - 2

Longest border of $p[1..j-1]$ equal
The KMP Algorithm - 3

The KMP Algorithm is a string searching algorithm that searches for occurrences of a "pattern" string within a "text" string, by employing the observation that when a mismatch occurs, the pattern itself embodies sufficient information for determining whether the next character examined needs to be compared. The algorithm therefore avoids reexamining text that has already been examined, resulting in a significant speedup over simple string searching algorithms.

The KMP algorithm uses a partial match table (also known as a failure function) to determine how much of the pattern can be skipped in case of a mismatch. This table is built up during the preprocessing phase and is used to guide the search process.

In the diagram, the text string $x$ and the pattern string $p$ are shown, with the process of the algorithm in progress. The algorithm compares characters at positions $i$ and $j$ in the text and pattern, respectively. If the characters match, the index $i$ is incremented. If they do not match, the algorithm uses the partial match table to determine how much of the pattern can be skipped before reattempting the comparison at a new position in the text.

The partial match table is computed such that if a mismatch occurs at position $i$ in the text, the algorithm can skip ahead in the pattern by at least the value stored in the table at position $i$. This allows the algorithm to avoid unnecessary comparisons and achieve linear time complexity on average.
Sunday Variant of Boyer-Moore

- A simplified version of the Boyer-Moore algorithm.
- Preprocessing: find the rightmost occurrence of each letter in $p$.
- Time complexity $O(mn)$.
- However, very fast in practice.
The BMS Algorithm - 1

\[
\begin{array}{ccccccc}
& & & 1 & i-m+1 & i & n \\
\hline
x & \ldots & \boxed{a} & \ldots \\
p & 1 & \boxed{b} & m \\
\end{array}
\]
The BMS Algorithm - 2

The diagram illustrates the BMS algorithm with a pattern `p` of length `m` and a text `x` of length `n`. The algorithm compares the pattern and the text characters to determine if the pattern is contained within the text. The diagram shows the alignment of the pattern within the text, with characters `a`, `b`, and `c` indicating match positions. The algorithm moves the pattern along the text, skipping unnecessary comparisons based on the alignment of characters.
Abstract

Introduction

Specific Patterns

Exact Pattern-Matching

Approximate Pattern-Matching

Skipping — KMP

Skipping — Sunday Variant of Boyer-Moore

Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)

Hybrid — FJS & SWY

The BMS Algorithm - 3

![Diagram of the BMS Algorithm](image-url)
The BMS Algorithm - 4

1  \ldots  d
x

1 m

1 \ldots d
p

1

n

Bill Smyth
Computing Patterns in Strings I
The BMS Algorithm - 5

Here is a diagram illustrating the BMS Algorithm, which is a variant of the Boyer-Moore algorithm. The algorithm aims to efficiently search for patterns within a text. The diagram shows a string `x` and a pattern `p` being matched. The idea is to shift the pattern to the right as much as possible, based on the last occurrence of characters within the pattern. This can significantly reduce the number of comparisons needed, making the search faster than straightforward string comparison.
The BMS Algorithm - 6

The diagram illustrates the BMS Algorithm for string matching. The text mentions various skipping techniques such as KMP, Sunday Variant of Boyer-Moore, and hybrid methods like FJS & SWY. The algorithm compares patterns and uses bit-parallel methods for efficiency.
The DBG Shift-Or Algorithm

- Makes use of the bit-parallel nature of computer words.
- Preprocessing: computes a bit array $B$ identifying each letter in $p$.
- Time complexity $O(mn/w)$, where $w$ is the computer word length.
- Fast for shorter patterns.
- Very flexible – easily modified for approximate matching.
## DBG Shift-Or

### Table: Pattern Preprocessing

<table>
<thead>
<tr>
<th>$p \setminus \Sigma$</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Algorithm:

- **Preprocess**($B$)
  - $s[0..m] \leftarrow 0(1^m)$ — initialize state vector
- **for** $i \leftarrow 1$ **to** $n$ **do**
  - $s \leftarrow \text{rightshift}(s, 1) \lor B[1..m, x[i]]$
  - **if** $s[m] = 0$ **then output** $i - m + 1$

At each step the state vector $s$ is shifted right one bit (0 enters from the left) and a logical OR is done with the column of $B$ corresponding to the current position $i$ in $x$. 

---

Bill Smyth  Computing Patterns in Strings I
Franek-Jennings-Smyth

- Combines KMP & BMS.
- So inherits the merits of both algorithms: very fast both asymptotically \(O(n)\) and in practice.

Here it is:

1. Perform **Sunday** shift along text.
2. When a match of letters is found at the end of the pattern, switch to **KMP** matching.
3. Continue **KMP** matching until no border can be used, then switch back to **Sunday** shift.
Combines BMS & Shift-Or.
Adapts to the nature of the text, thus on average faster than either component.

Here it is:

1. Perform **Sunday shift** along text.
2. When a match of letters is found at the end of the pattern, switch to **Shift-Or matching**.
3. Continue **Shift-Or** until no match can be found at the current position (the state vector $s$ is all ones), then skip to next possible position and switch back to **Sunday** shift.
Outline

1. Abstract
2. Introduction
   - What is a string?
   - Why are strings important?
   - Examples
   - String conferences
   - Important ideas
3. Computing Specific Patterns
4. Exact Pattern-Matching
   - Skipping — KMP
   - Skipping — Sunday Variant of Boyer-Moore
   - Bit-Parallel: Dömölki (1964), Baeza-Yates/Gonnet (1992)
   - Hybrid — FJS & SWY
5. Approximate Pattern-Matching
Overview

There are four main paradigms of approximate pattern-matching:

- pattern $p$ and text $x$ are well-defined; handled by dynamic programming in $O(mn)$, maybe $O(n \log m)$, time;
- letters in either $p$ or $x$ may be indeterminate; handled by modifications to exact pattern-matching algorithms (especially DBG & Sunday);
- letters are exact or indeterminate, with bounds given on both maximum and total distance (for example, $(\delta, \gamma)$-matching of musical texts in musical databases);
- match is exact but positions may be scrambled (Abelian matching); handled by convolutions.
Applications of Approximate Matching

- Recognition/correction of misspellings or word inversion in database/internet search.
- Tolerance of transcription errors in DNA sequences copied elsewhere in the genome.
- Appropriate handling of legitimate ambiguity, such as in protein/DNA entries, or in spelling variants (among dialects, or between past and present — for example, “itemise” and “itemize”, or Smyth and Smith).
- Matching of inherently approximate texts, such as musical passages or rhythms (to detect plagiarism, for example).
$p$ & $x$ Well-Defined

- Hamming distance (substitution only)
- Edit distance (plus insertion & deletion)
- Scoring distance (distinct scores for each pair of letters)

Usually a threshold $k$ is given; if the pattern $p$ is no more than distance $k$ from a substring $u = x[i..i+m-1]$, then $u$ is a $k$-match for $p$. 

Bill Smyth
Computing Patterns in Strings I
Hamming distance $d(p, x[2..5]) = 1$ implies that one substitution yields a match.
Using deletions and insertions can sometimes reduce the distance:

\[ x = abcd; \quad p = adbc. \]

Here the Hamming distance \( d(p, x) = 3 \), but deleting \( d \) from \( x \), then reinserting it after position 1 (two operations), yields edit distance \( d'(p, x) = 2 \).

More generally, scoring distance (often used in DNA analysis) gives different weights (or scores) to each operation — for example, reflecting the probability that letter \( A \) is deleted, or that \( C \to G \).

Pattern-matching using all of these forms of distance is implemented using dynamic programming.
Edit/Scoring distance II

\[ x = \text{m} \ i \ s \ s \ i \ s \ s \ i \ p \ p \ i \]
\[ p = \ i \ s \ i \]
\[ i \ s \ i \]

Edit distance \( d(p, x[2..5]) = 1 \) since deleting \( x[3] \) yields a match. (Also \( d(p, x[2..4]) = 1 \) by substituting \( x[4] \leftarrow i \), \( d(p, x[3..5]) = 1 \) by substituting \( x[3] \leftarrow i \).)
Indeterminate Pattern-Matching

This is common in applications to DNA/protein sequences, where a letter may legitimately take one of several values, and so match with each of them. For example, the string

\[ h\{a, i, o, u\}t \]

matches

\[ hat, \ hit, h\{o, u\}t. \]

A new area of research since 2003; the subject of Shu Wang’s Ph.D. dissertation. Bit-parallel and hybrid approaches are effective.
Abstract

Introduction

Specific Patterns

Exact Pattern-Matching

Approximate Pattern-Matching

\((\delta, \gamma)\)-Matching

In musical texts, the notes can be represented by integers. A match occurs if

\[
\max_{j=1}^{m} |p[j] - x[i+j-1]| \leq \delta,
\]

\[
\sum_{j=1}^{m} |p[j] - x[i+j-1]| \leq \gamma.
\]

Shift-Or is used, and modifications of other exact methods.
This is an even newer area of research, again motivated by applications in bioinformatics. In

\[ x = mississippi, \]

under Abelian matching, we find five matches with \( p = sis \)!