Efficient Variants of the
Backward-Oracle-Matching Algorithm

Simone Faro\textsuperscript{1} and Thierry Lecroq\textsuperscript{2}

\textsuperscript{1}Dipartimento di Matematica e Informatica, Università di Catania, Viale A.Doria n.6, 95125 Catania, Italy
\textsuperscript{2}Université de Rouen, LITIS EA 4108, 76821 Mont-Saint-Aignan Cedex, France

faro@DMI.UNICT.IT, thierry.lecroq@univ-rouen.fr
Abstract

In this article we present two efficient variants of the BOM string matching algorithm which are more efficient and flexible than the original algorithm. We also present bit-parallel versions of them obtaining an efficient variant of the BNDM algorithm. Then we compare the newly presented algorithms with some of the most recent and effective string matching algorithms. It turns out that the new proposed variants are very flexible and achieve very good results, especially in the case of large alphabets.
The String Matching Problem

Given a text $t$ of length $n$ and a pattern $p$ of length $m$ over some alphabet $\Sigma$ of size $\sigma$, the **string matching problem** consists in finding all occurrences of the pattern $p$ in the text $t$.

| t | a | c | c | a | b | a | c | c | b | a | a | b | b | a | c | c | b | a | a | b | c | a | a | c |

| p | c | c | b | a | a | b |
The String Matching Problem

Given a text $t$ of length $n$ and a pattern $p$ of length $m$ over some alphabet $\Sigma$ of size $\sigma$, the **string matching problem** consists in finding all occurrences of the pattern $p$ in the text $t$.
The String Matching Problem

Given a text $t$ of length $n$ and a pattern $p$ of length $m$ over some alphabet $\Sigma$ of size $\sigma$, the string matching problem consists in finding all occurrences of the pattern $p$ in the text $t$.

Applications:

- Image processing
- Signal processing
- Computational biology
- Computational chemistry
- Speech analysis
- Speech recognition
# The String Matching Problem

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force</td>
<td>Zhu-Takaoka</td>
</tr>
<tr>
<td>DFA</td>
<td>Berry-Ravindran</td>
</tr>
<tr>
<td>Karp-Rabin</td>
<td>Smith</td>
</tr>
<tr>
<td>Shift Or</td>
<td>Raita</td>
</tr>
<tr>
<td>Morris-Pratt</td>
<td>Reverse Factor</td>
</tr>
<tr>
<td>Knuth-Morris-Pratt</td>
<td>Turbo Reverse Factor</td>
</tr>
<tr>
<td>Simon</td>
<td>Forward Dawg Matching</td>
</tr>
<tr>
<td>Colussi</td>
<td>BOM</td>
</tr>
<tr>
<td>Galil-Giancarlo</td>
<td>Galil-Seiferas</td>
</tr>
<tr>
<td>Apostolico-Crochemore</td>
<td>Two Way</td>
</tr>
<tr>
<td>Not So Naive</td>
<td>Optimal Mismatch</td>
</tr>
<tr>
<td>Boyer-Moore</td>
<td>Maximal Shift</td>
</tr>
<tr>
<td>Turbo BM</td>
<td>Skip Search</td>
</tr>
<tr>
<td>Apostolico-Giancarlo</td>
<td>KMP Skip Search</td>
</tr>
<tr>
<td>Reverse Colussi</td>
<td>Alpha Skip Search</td>
</tr>
<tr>
<td>Horspool</td>
<td>Fast Search</td>
</tr>
<tr>
<td>Quick Search</td>
<td>Forward Fast Search</td>
</tr>
<tr>
<td>Tuned Boyer-Moore</td>
<td></td>
</tr>
</tbody>
</table>
The String Matching Problem

- Brute Force
- DFA
- Karp-Rabin
- Shift Or
- Morris-Pratt
- Knuth-Morris-Pratt
- Simon
- Colussi
- Galil-Giancarlo
- Apostolico-Crochemore
- Not So Naive
  - **Boyer-Moore**
  - Turbo BM
  - Apostolico-Giancarlo
  - Reverse Colussi
  - Horspool
  - Quick Search
  - Tuned Boyer-Moore
- Zhu-Takaoka
- Berry-Ravindran
- Smith
- Raita
- Reverse Factor
- Turbo Reverse Factor
- Forward Dawg Matching
- **BNDM**
- BOM
- Galil-Seiferas
- Two Way
- Optimal Mismatch
- Maximal Shift
- Skip Search
- KMP Skip Search
- Alpha Skip Search
- Fast Search
- Forward Fast Search
The String Matching Problem

Brute Force
DFA
Karp-Rabin
Shift Or
Morris-Pratt
Knuth-Morris-Pratt
Simon
Colussi
Galil-Giancarlo
Apostolico-Crochemore
Not So Naive
Boyer-Moore
Turbo BM
Apostolico-Giancarlo
Reverse Colussi
Horspool
Quick Search
Tuned Boyer-Moore

Zhu-Takaoka
Berry-Ravindran
Smith
Raita
Reverse Factor
Turbo Reverse Factor
Forward Dawg Matching
BNDM
BOM
Galil-Seiferas
Two Way
Optimal Mismatch
Maximal Shift
Skip Search
KMP Skip Search
Alpha Skip Search
Fast Search
Forward Fast Search
Automata

Automata based solutions have been also developed to design algorithms which have optimal sublinear performance on average. This is done by using factor automata, data structures which identify all factors of a word.

- **BOM** (Backward Oracle Matching) algorithm [2] is the most efficient, especially for long patterns.

- **BNDM** (Backward Nondeterministic Dawg Match) algorithm [3], is very efficient for short patterns.

---


Factor Automaton

The factor automaton of a pattern $p$, $\text{Aut}(p)$, is also called the factor DAWG of $p$ (for Directed Acyclic Word Graph). Such an automaton recognizes all the factors of $p$. Formally the language recognized by $\text{Aut}(p)$ is defined as follows

$$\mathcal{L}(\text{Aut}(p)) = \{u \in \Sigma^* : \text{ exists } v, w \in \Sigma^* \text{ such that } p = vwv\}.$$
Factor Automaton

The factor automaton of a pattern \( p \), \( Aut(p) \), is also called the factor DAWG of \( p \) (for Directed Acyclic Word Graph). Such an automaton recognizes all the factors of \( p \). Formally the language recognized by \( Aut(p) \) is defined as follows

\[
\mathcal{L}(Aut(p)) = \{ u \in \Sigma^* : \text{exists } v, w \in \Sigma^* \text{ such that } p = vwv \}.
\]

The factor automaton of the pattern \( p = abbbaab \)
Factor Oracle

The factor oracle of a pattern $p$, Oracle($p$), is a very compact automaton which recognizes at least all the factors of $p$ and slightly more other words. Formally Oracle($p$) is an automaton $\{Q, m, Q, \Sigma, \delta\}$ such that

1. $Q$ contains exactly $m + 1$ states, say $Q = \{0, 1, 2, 3, \ldots, m\}$
2. $m$ is the initial state
3. all states are final
4. the language accepted by Oracle($p$) is such that $L(\text{Aut}(p)) \subseteq L(\text{Oracle}(p))$
The factor oracle of a pattern \( p \), \( \text{Oracle}(p) \), is a very compact automaton which recognizes at least all the factors of \( p \) and slightly more other words. Formally \( \text{Oracle}(p) \) is an automaton \( \{ Q, m, Q, \Sigma, \delta \} \) such that

1. \( Q \) contains exactly \( m + 1 \) states, say \( Q = \{ 0, 1, 2, 3, \ldots, m \} \)
2. \( m \) is the initial state
3. all states are final
4. the language accepted by \( \text{Oracle}(p) \) is such that \( L(\text{Aut}(p)) \subseteq L(\text{Oracle}(p)) \)

The factor automaton of the pattern \( p = abbbaab \)
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.

\[
t = \begin{array}{ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of \( p \) and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter $c$ after the reading of a word $u$ then $cu$ is not a factor of $p$ and moving the beginning of the window just after $c$ is secure. If a factor of length $m$ is recognized then we have found an occurrence of the pattern.
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.

```
t  a  c  c  a  a  a  b  b  a  a  b  b  b  a  a  c  b  a  a  b  c  a  a  c
```
Bacward Oracle Matching

The basic idea of the BDM and BOM algorithms is that if its backward search failed on a letter c after the reading of a word u then cu is not a factor of p and moving the beginning of the window just after c is secure. If a factor of length m is recognized then we have found an occurrence of the pattern.
Extending BOM

Now we present an extension of the BOM algorithm by introducing a fast-loop with the aim of obtaining better results on the average. We discuss the application of different variations of the fast-loop and present experimental results in order to identify the best choice.
The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

Generally a fast-loop is implemented by iterating the bad character heuristic in a checkless cycle, in order to quickly locate an occurrence of the rightmost character of the pattern.

\[ bc(c) = \min(\{0 \leq k < m \mid p[m - 1 - k] = c\} \cup \{m\}) \]

\[ \begin{align*}
    k &= bc(t_j) \\
    \text{while} & \ (k \neq 0) \ \text{do} \\
    j &= j + k \\
    k &= bc(t_j)
\end{align*} \]
The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm \[4\] and later largely used in almost all variations of the Boyer-Moore algorithm.

\[
\begin{array}{cccccccccc}
t & a & c & c & a & a & a & b & b & a & a & b & c & b & a & a & c \\
p & a & b & a & c & c \\
\end{array}
\]

(A)

\[
\begin{align*}
k &= bc(t_j) \\
\text{while } (k \neq 0) \text{ do} \\
j &= j + k \\
k &= bc(t_j)
\end{align*}
\]

The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

\[
\begin{array}{cccccccccccccc}
\text{t} & a & c & c & a & a & a & b & b & a & a & b & c & b & a & a & c \\
\text{p} & a & b & & a & & c & & c
\end{array}
\]

\[(A)\]

\[
k = bc(t_j) \\
\text{while } (k \neq 0) \text{ do} \\
\quad j = j + k \\
\quad k = bc(t_j)
\]

The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

\[ k = bc(t_j) \]
\[ \text{while } (k \neq 0) \text{ do} \]
\[ j = j + k \]
\[ k = bc(t_j) \]
The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

```
<table>
<thead>
<tr>
<th>t</th>
<th>a</th>
<th>c</th>
<th>c</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

(A)

\[
k = bc(t_j)
\]

while (k ≠ 0) do
  \[
j = j + k
\]
  \[
k = bc(t_j)
\]
The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

\[
\begin{align*}
\text{t} & \quad \text{a} \quad \text{c} \quad \text{c} \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{c} \quad \text{b} \quad \text{a} \quad \text{a} \quad \text{c} \\
\text{p} & \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{c} \quad \text{c} \\
\end{align*}
\]

(A)

\[
\begin{align*}
k &= bc(t_j) \\
\text{while} \ (k \neq 0) \ &\text{do} \\
&\quad j = j + k \\
&\quad k = bc(t_j)
\end{align*}
\]

The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

\[
\begin{array}{ccccccccccc}
\text{t} & a & c & c & a & a & a & b & b & a & a & b & c & b & a & a & c \\
\text{p} & a & b & a & c & c
\end{array}
\]

(A)

\[
k = bc(t_j) \\
\text{while } (k \neq 0) \text{ do} \\
\quad j = j + k \\
\quad k = bc(t_j)
\]

The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

\[
\begin{align*}
t & \quad a \quad c \quad c \quad a \quad a \quad a \quad b \quad b \quad b \quad a \quad a \quad b \quad c \quad b \quad a \quad a \quad a \quad c \\
p & \quad \framebox{a} \quad \framebox{b} \quad \framebox{a} \quad c \quad c \\
\end{align*}
\]

\[(A)\]

\[
\begin{align*}
k &= bc(t_j) \\
\text{while } (k \neq 0) \text{ do} \\
&\hspace{1cm} j = j + k \\
&\hspace{1cm} k = bc(t_j)
\end{align*}
\]

The classical fast-loop

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm [4] and later largely used in almost all variations of the Boyer-Moore algorithm.

\textbf{t} \hspace{1cm} a \hspace{0.2cm} c \hspace{0.2cm} c \hspace{0.2cm} a \hspace{0.2cm} a \hspace{0.2cm} a \hspace{0.2cm} b \hspace{0.2cm} b \hspace{0.2cm} a \hspace{0.2cm} a \hspace{0.2cm} b \hspace{0.2cm} c \hspace{0.2cm} b \hspace{0.2cm} a \hspace{0.2cm} a \hspace{0.2cm} a \hspace{0.2cm} c

\textbf{p} \hspace{1cm} a \hspace{0.2cm} b \hspace{0.2cm} a \hspace{0.2cm} c \hspace{0.2cm} c

(A)

\[ k = bc(t_j) \]
\[ \text{while } (k \neq 0) \text{ do} \]
\[ j = j + k \]
\[ k = bc(t_j) \]

The classical fast-loop has first introduced in the Tuned-Boyer-Moore algorithm \cite{4} and later largely used in almost all variations of the Boyer-Moore algorithm.

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>Experimental results with $\sigma = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>157.62</td>
<td>95.95</td>
</tr>
<tr>
<td>8</td>
<td>85.48</td>
<td>58.66</td>
</tr>
<tr>
<td>16</td>
<td>43.04</td>
<td>43.36</td>
</tr>
<tr>
<td>32</td>
<td>26.63</td>
<td>35.00</td>
</tr>
<tr>
<td>64</td>
<td>17.39</td>
<td>28.05</td>
</tr>
<tr>
<td>128</td>
<td>15.28</td>
<td>23.68</td>
</tr>
<tr>
<td>256</td>
<td>10.79</td>
<td>19.86</td>
</tr>
<tr>
<td>512</td>
<td>6.18</td>
<td>14.29</td>
</tr>
<tr>
<td>1024</td>
<td>3.29</td>
<td>8.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>Experimental results with $\sigma = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>103.28</td>
<td>66.81</td>
</tr>
<tr>
<td>8</td>
<td>71.59</td>
<td>38.72</td>
</tr>
<tr>
<td>16</td>
<td>39.61</td>
<td>26.57</td>
</tr>
<tr>
<td>32</td>
<td>18.68</td>
<td>21.80</td>
</tr>
<tr>
<td>64</td>
<td>12.67</td>
<td>20.09</td>
</tr>
<tr>
<td>128</td>
<td>14.22</td>
<td>19.38</td>
</tr>
<tr>
<td>256</td>
<td>8.81</td>
<td>19.05</td>
</tr>
<tr>
<td>512</td>
<td>4.62</td>
<td>17.73</td>
</tr>
<tr>
<td>1024</td>
<td>2.35</td>
<td>11.49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>Experimental results with $\sigma = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>78.76</td>
<td>55.23</td>
</tr>
<tr>
<td>8</td>
<td>51.68</td>
<td>30.37</td>
</tr>
<tr>
<td>16</td>
<td>35.40</td>
<td>19.92</td>
</tr>
<tr>
<td>32</td>
<td>20.62</td>
<td>16.12</td>
</tr>
<tr>
<td>64</td>
<td>12.11</td>
<td>14.84</td>
</tr>
<tr>
<td>128</td>
<td>12.60</td>
<td>15.63</td>
</tr>
<tr>
<td>256</td>
<td>7.58</td>
<td>16.73</td>
</tr>
<tr>
<td>512</td>
<td>4.29</td>
<td>17.90</td>
</tr>
<tr>
<td>1024</td>
<td>2.87</td>
<td>14.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>Experimental results with $\sigma = 64$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>64.84</td>
<td>50.93</td>
</tr>
<tr>
<td>8</td>
<td>39.35</td>
<td>27.44</td>
</tr>
<tr>
<td>16</td>
<td>26.09</td>
<td>17.12</td>
</tr>
<tr>
<td>32</td>
<td>19.45</td>
<td>14.09</td>
</tr>
<tr>
<td>64</td>
<td>13.15</td>
<td>13.58</td>
</tr>
<tr>
<td>128</td>
<td>13.11</td>
<td>17.67</td>
</tr>
<tr>
<td>256</td>
<td>6.25</td>
<td>18.04</td>
</tr>
<tr>
<td>512</td>
<td>2.91</td>
<td>18.00</td>
</tr>
<tr>
<td>1024</td>
<td>2.71</td>
<td>16.89</td>
</tr>
</tbody>
</table>

\begin{align*}
    k &= bc(t_j) \\
    \text{while} \ (k \neq 0) \text{ do} \\
    \quad j &= j + k \\
    k &= bc(t_j)
\end{align*}

A fast-loop over transitions

We can translate the idea of the fast-loop over automaton transitions. This consists in shifting the pattern along the text with no more check until a non-undefined transition is found with the rightmost character of the current window of the text.

\[
\begin{align*}
(B) \\
q &= \delta(m, t_j) \\
\textbf{while} \ (q == \bot) \ \textbf{do} \\
& \quad j = j + m \\
& \quad q = \delta(m, t_j)
\end{align*}
\]
A fast-loop over transitions

We can translate the idea of the fast-loop over automaton transitions. This consists in shifting the pattern along the text with no more check until a non-undefined transition is found with the rightmost character of the current window of the text.
A fast-loop over transitions

We can translate the idea of the fast-loop over automaton transitions. This consists in shifting the pattern along the text with no more check until a non-undefined transition is found with the rightmost character of the current window of the text.

\[
q = \delta(m, t_j) \\
\textbf{while} (q == \perp) \textbf{ do} \\
\quad j = j + m \\
\quad q = \delta(m, t_j)
\]
A fast-loop over transitions

We can translate the idea of the fast-loop over automaton transitions. This consists in shifting the pattern along the text with no more check until a non-undefined transition is found with the rightmost character of the current window of the text.

\[
\begin{align*}
q &= \delta(m, t_j) \\
\text{while} \ (q == \bot) \ \text{do} \\
& \quad j = j + m \\
&q = \delta(m, t_j)
\end{align*}
\]
A fast-loop over transitions

We can translate the idea of the fast-loop over automaton transitions. This consists in shifting the pattern along the text with no more check until a non-undefined transition is found with the rightmost character of the current window of the text.

\[
\begin{align*}
q &= \delta(m, t_j) \\
\text{while } (q == \bot) \text{ do} \\
& \quad j = j + m \\
& \quad q = \delta(m, t_j)
\end{align*}
\]
A fast-loop over transitions

We can translate the idea of the fast-loop over automaton transitions. This consists in shifting the pattern along the text with no more check until a non-undefined transition is found with the rightmost character of the current window of the text.

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>BOM (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>157.62</td>
<td>95.95</td>
</tr>
<tr>
<td>8</td>
<td>85.48</td>
<td>58.66</td>
</tr>
<tr>
<td>16</td>
<td>43.04</td>
<td>43.36</td>
</tr>
<tr>
<td>32</td>
<td>26.63</td>
<td>35.00</td>
</tr>
<tr>
<td>64</td>
<td>17.39</td>
<td>28.05</td>
</tr>
<tr>
<td>128</td>
<td>15.28</td>
<td>23.68</td>
</tr>
<tr>
<td>256</td>
<td>10.79</td>
<td>19.86</td>
</tr>
<tr>
<td>512</td>
<td>6.18</td>
<td>14.29</td>
</tr>
<tr>
<td>1024</td>
<td>3.29</td>
<td>8.20</td>
</tr>
</tbody>
</table>

Experimental results with $\sigma = 8$

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>BOM (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>103.28</td>
<td>66.81</td>
</tr>
<tr>
<td>8</td>
<td>71.59</td>
<td>38.72</td>
</tr>
<tr>
<td>16</td>
<td>39.61</td>
<td>26.57</td>
</tr>
<tr>
<td>32</td>
<td>18.68</td>
<td>21.80</td>
</tr>
<tr>
<td>64</td>
<td>12.67</td>
<td>20.09</td>
</tr>
<tr>
<td>128</td>
<td>14.22</td>
<td>19.38</td>
</tr>
<tr>
<td>256</td>
<td>8.81</td>
<td>19.05</td>
</tr>
<tr>
<td>512</td>
<td>4.62</td>
<td>17.73</td>
</tr>
<tr>
<td>1024</td>
<td>2.35</td>
<td>11.49</td>
</tr>
</tbody>
</table>

Experimental results with $\sigma = 16$

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>BOM (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>64.84</td>
<td>50.93</td>
</tr>
<tr>
<td>8</td>
<td>39.35</td>
<td>27.44</td>
</tr>
<tr>
<td>16</td>
<td>26.09</td>
<td>17.12</td>
</tr>
<tr>
<td>32</td>
<td>19.45</td>
<td>14.09</td>
</tr>
<tr>
<td>64</td>
<td>13.15</td>
<td>13.58</td>
</tr>
<tr>
<td>128</td>
<td>13.11</td>
<td>17.67</td>
</tr>
<tr>
<td>256</td>
<td>6.25</td>
<td>18.04</td>
</tr>
<tr>
<td>512</td>
<td>2.91</td>
<td>18.00</td>
</tr>
<tr>
<td>1024</td>
<td>2.71</td>
<td>16.89</td>
</tr>
</tbody>
</table>

Experimental results with $\sigma = 32$

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>BOM (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>13.11</td>
<td>17.67</td>
</tr>
<tr>
<td>8</td>
<td>13.15</td>
<td>13.58</td>
</tr>
<tr>
<td>16</td>
<td>26.09</td>
<td>17.12</td>
</tr>
<tr>
<td>32</td>
<td>19.45</td>
<td>14.09</td>
</tr>
<tr>
<td>64</td>
<td>13.11</td>
<td>17.67</td>
</tr>
<tr>
<td>128</td>
<td>6.25</td>
<td>18.04</td>
</tr>
<tr>
<td>256</td>
<td>2.91</td>
<td>18.00</td>
</tr>
<tr>
<td>512</td>
<td>2.71</td>
<td>16.89</td>
</tr>
<tr>
<td>1024</td>
<td>2.71</td>
<td>16.89</td>
</tr>
</tbody>
</table>

Experimental results with $\sigma = 64$

<table>
<thead>
<tr>
<th>$m$</th>
<th>BOM (A)</th>
<th>BOM (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6.25</td>
<td>18.04</td>
</tr>
<tr>
<td>8</td>
<td>2.91</td>
<td>18.00</td>
</tr>
<tr>
<td>16</td>
<td>2.71</td>
<td>16.89</td>
</tr>
<tr>
<td>32</td>
<td>2.71</td>
<td>16.89</td>
</tr>
<tr>
<td>64</td>
<td>2.71</td>
<td>16.89</td>
</tr>
<tr>
<td>128</td>
<td>2.71</td>
<td>16.89</td>
</tr>
<tr>
<td>256</td>
<td>2.71</td>
<td>16.89</td>
</tr>
<tr>
<td>512</td>
<td>2.71</td>
<td>16.89</td>
</tr>
</tbody>
</table>

(B)

\[ q = \delta(m, t_j) \]

while \( q = \bot \) do

\[ j = j + m \]

\[ q = \delta(m, t_j) \]
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.

```
(C)
q = δ(m, t_j)
if q ≠ ⊥ then
  p = δ(q, t_{j-1})
while (p == ⊥) do
  j = j + m - 1
  q = δ(m, t_j)
  if q ≠ ⊥ then
    p = δ(q, t_{j-1})
```
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.

\[
\begin{align*}
t & = a c c a a c a b a a b a b a b a a c \\
p & = b a a b b b a \\
\end{align*}
\]

(C)

\[
\begin{align*}
q &= \delta(m, t_j) \\
\text{if } q \neq \bot \text{ then } \\
p &= \delta(q, t_{j-1}) \\
\text{while } (p == \bot) \text{ do } \\
j &= j + m - 1 \\
q &= \delta(m, t_j) \\
\text{if } q \neq \bot \text{ then } \\
p &= \delta(q, t_{j-1})
\end{align*}
\]
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.

\[
\begin{align*}
\text{t} & \quad a \quad c \quad c \quad a \quad a \quad c \quad a \quad b \quad a \quad a \quad b \quad a \quad b \quad a \quad a \quad c \\
\text{p} & \quad b \quad a \quad a \quad b \quad b \quad b \quad b \quad a
\end{align*}
\]

\[
\begin{align*}
\text{(C)} & \\
q &= \delta(m, t_j) \\
\text{if } q \neq \bot \text{ then} & \\
p &= \delta(q, t_{j-1}) \\
\text{while } (p \neq \bot) \text{ do} & \\
j &= j + m - 1 \\
q &= \delta(m, t_j) \\
\text{if } q \neq \bot \text{ then} & \\
p &= \delta(q, t_{j-1})
\end{align*}
\]
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.

(C)

\[ q = \delta(m, t_j) \]

if \( q \neq \perp \) then

\[ p = \delta(q, t_{j-1}) \]

while \( p = \perp \) do

\[ j = j + m - 1 \]

\[ q = \delta(m, t_j) \]

if \( q \neq \perp \) then

\[ p = \delta(q, t_{j-1}) \]
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.

\[
q = \delta(m, t_j)
\]
\[
\text{if } q \neq \bot \text{ then}
\]
\[
p = \delta(q, t_{j-1})
\]
\[
\text{while } (p == \bot) \text{ do}
\]
\[
j = j + m - 1
\]
\[
q = \delta(m, t_j)
\]
\[
\text{if } q \neq \bot \text{ then}
\]
\[
p = \delta(q, t_{j-1})
\]
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.

\[ q = \delta(m, t_j) \]
\[ \text{if } q \neq \bot \text{ then} \]
\[ p = \delta(q, t_{j-1}) \]
\[ \text{while } (p == \bot) \text{ do} \]
\[ j = j + m - 1 \]
\[ q = \delta(m, t_j) \]
\[ \text{if } q \neq \bot \text{ then} \]
\[ p = \delta(q, t_{j-1}) \]
A fast-loop over two transitions

The variation of the algorithm we propose tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition.

\[
q = \delta(m, t_j) \\
\text{if } q \neq \bot \text{ then} \\
p = \delta(q, t_{j-1}) \\
\text{while } (p == \bot) \text{ do} \\
j = j + m - 1 \\
q = \delta(m, t_j) \\
\text{if } q \neq \bot \text{ then} \\
p = \delta(q, t_{j-1})
\]

| Experimental results with $\sigma = 8$ | Experimental results with $\sigma = 16$ |
|---|---|---|---|---|---|---|---|
| $m$ | BOM | (A) | (B) | (C) | $m$ | BOM | (A) | (B) | (C) |
| 4  | 157.62 | 95.95 | 135.95 | 109.03 | 4  | 103.28 | 66.81 | 86.28 | 93.53 |
| 8  | 85.48  | 58.66 | 78.70  | 58.63  | 8  | 71.59  | 38.72 | 60.27  | 44.02  |
| 16 | 43.04  | 43.36 | 43.00  | 37.15  | 16 | 39.61  | 26.57 | 35.70  | 23.68  |
| 32 | 26.63  | 35.00 | 28.29  | 25.93  | 32 | 18.68  | 21.80 | 18.82  | 15.71  |
| 64 | 17.39  | 28.05 | 17.13  | 17.00  | 64 | 12.67  | 20.09 | 12.55  | 12.73  |
| 256| 10.79  | 19.86 | 9.60   | 9.76   | 256| 8.81   | 19.05 | 8.12   | 7.83   |
| 512| 6.18   | 14.29 | 6.11   | 5.76   | 512| 4.62   | 17.73 | 4.62   | 4.53   |
| 1024| 3.29  | 8.20 | 3.45   | 3.35   | 1024| 2.35  | 11.49 | 2.66   | 2.89   |

| Experimental results with $\sigma = 32$ | Experimental results with $\sigma = 64$ |
|---|---|---|---|---|---|---|---|
| $m$ | BOM | (A) | (B) | (C) | $m$ | BOM | (A) | (B) | (C) |
| 4  | 78.76 | 55.23 | 57.75 | 88.57 | 4  | 64.84 | 50.93 | 42.34 | 88.52 |
| 8  | 51.68 | 30.37 | 42.03 | 39.84 | 8  | 39.35 | 27.44 | 29.29 | 38.84 |
| 16 | 35.40 | 19.92 | 30.18 | 20.34 | 16 | 26.09 | 17.12 | 22.03 | 20.07 |
| 64 | 12.11 | 14.84 | 11.55 | 10.63 | 64 | 13.15 | 13.58 | 12.28 | 10.37 |
| 128| 12.60 | 15.63 | 11.26 | 10.01 | 128| 13.11 | 17.67 | 10.86 | 9.76  |
| 256| 7.58  | 16.73 | 6.32  | 5.90  | 256| 6.25  | 18.04 | 5.79  | 5.55  |
| 512| 4.29  | 17.90 | 3.73  | 3.83  | 512| 2.91  | 18.00 | 3.12  | 5.32  |
| 1024| 2.87 | 14.19 | 2.67 | 2.79 | 1024| 2.71 | 16.89 | 2.58 | 2.42 |
A fast-loop over two transitions

We could encapsulate the two first transitions of the oracle in the function

\[
\lambda(a, b) = \begin{cases} 
\bot & \text{if } \delta(m, a) = \bot \\
\delta(\delta(m, a), b) & \text{otherwise.}
\end{cases}
\]

(D)

\begin{align*}
q &= \lambda(t_j, t_{j-1}) \\
\textbf{while } (q == \bot) \textbf{ do} \\
    j &= j + m - 1 \\
    q &= \lambda(t_j, t_{j-1})
\end{align*}
A fast-loop over two transitions

We could encapsulate the two first transitions of the oracle in the function

\[
\lambda(a, b) = \begin{cases} 
\bot & \text{if } \delta(m, a) = \bot \\
\delta(\delta(m, a), b) & \text{otherwise.}
\end{cases}
\]

(D)

\[
q = \lambda(t_j, t_{j-1}) \\
\text{while } (q == \bot) \text{ do} \\
j = j + m - 1 \\
q = \lambda(t_j, t_{j-1})
\]
A fast-loop over two transitions

We could encapsulate the two first transitions of the oracle in the function

\[
\lambda(a, b) = \begin{cases} 
\bot & \text{if } \delta(m, a) = \bot \\
\delta(\delta(m, a), b) & \text{otherwise.}
\end{cases}
\]

\(\lambda\)

\[
\begin{array}{cccccccccc}
t & a & c & c & a & a & a & c & a & b & a & a & b & a & b & a & a & a & c \\
p & b & a & a & b & b & b & b & a \\
\lambda & a & b & c \\
a & 1 & 5 & \bot \\
b & 2 & 4 & \bot \\
c & \bot & \bot & \bot
\end{array}
\]
A fast-loop over two transitions

We could encapsulate the two first transitions of the oracle in the function

\[ \lambda(a, b) = \begin{cases} \bot & \text{if } \delta(m, a) = \bot \\ \delta(\delta(m, a), b) & \text{otherwise.} \end{cases} \]

\[ (D) \]

\[ q = \lambda(t_j, t_{j-1}) \]

\[ \text{while } (q == \bot) \text{ do} \]

\[ j = j + m - 1 \]

\[ q = \lambda(t_j, t_{j-1}) \]
A fast-loop over two transitions

We could encapsulate the two first transitions of the oracle in the function

$$\lambda(a, b) = \begin{cases} \perp & \text{if } \delta(m, a) = \perp \\ \delta(\delta(m, a), b) & \text{otherwise.} \end{cases}$$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>5</td>
<td>⊥</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>4</td>
<td>⊥</td>
</tr>
<tr>
<td>c</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

(D)

```
q = λ(t_j, t_{j-1})
while (q == ⊥) do
    j = j + m - 1
    q = λ(t_j, t_{j-1})
```
A fast-loop over two transitions

We could encapsulate the two first transitions of the oracle in the function

\[
\lambda(a, b) = \begin{cases} 
\bot & \text{if } \delta(m, a) = \bot \\
\delta(\delta(m, a), b) & \text{otherwise.}
\end{cases}
\]

\[
\begin{array}{cccccccccccc}
t & a & c & c & a & a & c & a & b & a & a & b & a & b & a & a & c \\
\end{array}
\]

\[
\begin{array}{ccccccccccc}
p & b & a & a & b & b & b & b & a \\
\end{array}
\]

\[
\begin{array}{ccc}
\lambda & a & b & c \\
a & 1 & 5 & \bot \\
b & 2 & 4 & \bot \\
c & \bot & \bot & \bot \\
\end{array}
\]

\[
(D) \\
q = \lambda(t_j, t_{j-1}) \\
\text{while } (q == \bot) \textbf{ do} \\
\quad j = j + m - 1 \\
\quad q = \lambda(t_j, t_{j-1})
\]
A fast-loop over two transitions

We could encapsulate the two first transitions of the oracle in the function

\[ \lambda(a, b) = \begin{cases} \bot & \text{if } \delta(m, a) = \bot \\ \delta(\delta(m, a), b) & \text{otherwise.} \end{cases} \]

Experimental results with \( \sigma = 8 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>BOM</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>157.62</td>
<td>95.95</td>
<td>135.95</td>
<td>109.03</td>
<td>55.35</td>
</tr>
<tr>
<td>8</td>
<td>85.48</td>
<td>58.66</td>
<td>78.70</td>
<td>58.63</td>
<td>34.16</td>
</tr>
<tr>
<td>16</td>
<td>43.04</td>
<td>43.36</td>
<td>43.00</td>
<td>37.15</td>
<td>26.82</td>
</tr>
<tr>
<td>32</td>
<td>26.63</td>
<td>35.00</td>
<td>28.29</td>
<td>25.93</td>
<td>21.25</td>
</tr>
<tr>
<td>64</td>
<td>17.39</td>
<td>28.05</td>
<td>17.13</td>
<td>17.00</td>
<td>14.42</td>
</tr>
<tr>
<td>128</td>
<td>15.28</td>
<td>23.68</td>
<td>15.75</td>
<td>15.87</td>
<td>12.87</td>
</tr>
<tr>
<td>256</td>
<td>10.79</td>
<td>19.86</td>
<td>9.60</td>
<td>9.76</td>
<td>8.53</td>
</tr>
<tr>
<td>512</td>
<td>6.18</td>
<td>14.29</td>
<td>6.11</td>
<td>5.76</td>
<td>4.76</td>
</tr>
<tr>
<td>1024</td>
<td>3.29</td>
<td>8.20</td>
<td>3.45</td>
<td>3.35</td>
<td>2.64</td>
</tr>
</tbody>
</table>

Experimental results with \( \sigma = 16 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>BOM</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>103.28</td>
<td>66.81</td>
<td>86.28</td>
<td>93.53</td>
<td>40.63</td>
</tr>
<tr>
<td>8</td>
<td>71.59</td>
<td>38.72</td>
<td>60.27</td>
<td>44.02</td>
<td>21.73</td>
</tr>
<tr>
<td>16</td>
<td>39.61</td>
<td>26.57</td>
<td>35.70</td>
<td>23.68</td>
<td>14.91</td>
</tr>
<tr>
<td>32</td>
<td>18.68</td>
<td>21.80</td>
<td>18.82</td>
<td>15.71</td>
<td>12.73</td>
</tr>
<tr>
<td>64</td>
<td>12.67</td>
<td>20.09</td>
<td>12.55</td>
<td>12.73</td>
<td>12.49</td>
</tr>
<tr>
<td>256</td>
<td>8.81</td>
<td>19.05</td>
<td>8.12</td>
<td>7.83</td>
<td>6.88</td>
</tr>
<tr>
<td>512</td>
<td>4.62</td>
<td>17.73</td>
<td>4.62</td>
<td>4.53</td>
<td>3.60</td>
</tr>
<tr>
<td>1024</td>
<td>2.35</td>
<td>11.49</td>
<td>2.66</td>
<td>2.89</td>
<td>2.67</td>
</tr>
</tbody>
</table>

Experimental results with \( \sigma = 32 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>BOM</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>78.76</td>
<td>55.23</td>
<td>57.75</td>
<td>88.57</td>
<td>37.44</td>
</tr>
<tr>
<td>8</td>
<td>51.68</td>
<td>30.37</td>
<td>42.03</td>
<td>39.84</td>
<td>18.59</td>
</tr>
<tr>
<td>16</td>
<td>35.40</td>
<td>19.92</td>
<td>30.18</td>
<td>20.34</td>
<td>12.29</td>
</tr>
<tr>
<td>32</td>
<td>20.62</td>
<td>16.12</td>
<td>19.34</td>
<td>12.20</td>
<td>11.58</td>
</tr>
<tr>
<td>64</td>
<td>12.11</td>
<td>14.84</td>
<td>11.55</td>
<td>10.63</td>
<td>11.10</td>
</tr>
<tr>
<td>128</td>
<td>12.60</td>
<td>15.63</td>
<td>11.26</td>
<td>10.01</td>
<td>7.46</td>
</tr>
<tr>
<td>256</td>
<td>7.58</td>
<td>16.73</td>
<td>6.32</td>
<td>5.90</td>
<td>3.79</td>
</tr>
<tr>
<td>512</td>
<td>4.29</td>
<td>17.90</td>
<td>3.73</td>
<td>3.83</td>
<td>3.20</td>
</tr>
<tr>
<td>1024</td>
<td>2.87</td>
<td>14.19</td>
<td>2.67</td>
<td>2.79</td>
<td>2.01</td>
</tr>
</tbody>
</table>

Experimental results with \( \sigma = 64 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>BOM</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>64.84</td>
<td>50.93</td>
<td>42.34</td>
<td>88.52</td>
<td>37.04</td>
</tr>
<tr>
<td>8</td>
<td>39.35</td>
<td>27.44</td>
<td>29.29</td>
<td>38.84</td>
<td>17.99</td>
</tr>
<tr>
<td>16</td>
<td>26.09</td>
<td>17.12</td>
<td>22.03</td>
<td>20.07</td>
<td>11.57</td>
</tr>
<tr>
<td>32</td>
<td>19.45</td>
<td>14.09</td>
<td>17.11</td>
<td>11.81</td>
<td>10.76</td>
</tr>
<tr>
<td>64</td>
<td>13.15</td>
<td>13.58</td>
<td>12.28</td>
<td>10.37</td>
<td>10.70</td>
</tr>
<tr>
<td>128</td>
<td>13.11</td>
<td>17.67</td>
<td>10.86</td>
<td>9.76</td>
<td>6.35</td>
</tr>
<tr>
<td>256</td>
<td>6.25</td>
<td>18.04</td>
<td>5.79</td>
<td>5.55</td>
<td>3.60</td>
</tr>
<tr>
<td>512</td>
<td>2.91</td>
<td>18.00</td>
<td>3.12</td>
<td>5.32</td>
<td>1.98</td>
</tr>
<tr>
<td>1024</td>
<td>2.71</td>
<td>16.89</td>
<td>2.58</td>
<td>2.42</td>
<td>1.57</td>
</tr>
</tbody>
</table>

(D)

\[ q = \lambda(t_j, t_{j-1}) \]
\[ \text{while } (q == \bot) \text{ do} \]
\[ j = j + m - 1 \]
\[ q = \lambda(t_j, t_{j-1}) \]
EXTENDED-BOM\((p, m, t, n)\)

1. \(\delta \leftarrow \text{precompute-factor-oracle}(p)\)
2. \(\textbf{for } a \in \Sigma \textbf{ do}\)
3. \(q \leftarrow \delta(m, a)\)
4. \(\textbf{for } b \in \Sigma \textbf{ do}\)
5. \(\text{if } q = \bot \textbf{ then } \lambda(a, b) \leftarrow \bot\)
6. \(\text{else } \lambda(a, b) \leftarrow \delta(q, b)\)
7. \(t[n .. n + m - 1] \leftarrow p\)
8. \(j \leftarrow m - 1\)
9. \(\textbf{while } j < n \textbf{ do}\)
10. \(q \leftarrow \lambda(t[j], t[j - 1])\)
11. \(\textbf{while } q = \bot \textbf{ do}\)
12. \(j \leftarrow j + m - 1\)
13. \(q \leftarrow \lambda(t[j], t[j - 1])\)
14. \(i \leftarrow j - 2\)
15. \(\textbf{while } q \neq \bot \textbf{ do}\)
16. \(q \leftarrow \delta(q, t[i])\)
17. \(i \leftarrow i - 1\)
18. \(\textbf{if } i < j - m + 1 \textbf{ then}\)
19. \(\text{output}(j)\)
20. \(i \leftarrow 1 + 1\)
21. \(j \leftarrow j + i + m\)
**Extended-BOM** \((p, m, t, n)\)

1. \(\delta \leftarrow \text{precompute-factor-oracle}(p)\)
2. \(\text{for } a \in \Sigma \text{ do}\)
   3. \(q \leftarrow \delta(m, a)\)
   4. \(\text{for } b \in \Sigma \text{ do}\)
      5. \(\text{if } q = \bot \text{ then } \lambda(a, b) \leftarrow \bot\)
      6. \(\text{else } \lambda(a, b) \leftarrow \delta(q, b)\)
7. \(t[n..n+m-1] \leftarrow p\)
8. \(j \leftarrow m-1\)

9. \(\text{while } j < n \text{ do}\)
   10. \(q \leftarrow \lambda(t[j], t[j-1])\)
   11. \(\text{while } q = \bot \text{ do}\)
   12. \(j \leftarrow j + m - 1\)
   13. \(q \leftarrow \lambda(t[j], t[j-1])\)
   14. \(i \leftarrow j - 2\)
   15. \(\text{while } q \neq \bot \text{ do}\)
   16. \(q \leftarrow \delta(q, t[i])\)
   17. \(i \leftarrow i - 1\)
   18. \(\text{if } i < j - m + 1 \text{ then}\)
   19. \(\text{output}(j)\)
   20. \(i \leftarrow i + 1\)
21. \(j \leftarrow j + i + m\)
**Extended-BOM** ($p, m, t, n$)

1. $\delta \leftarrow \text{precompute-factor-oracle}(p)$
2. \(\text{for } a \in \Sigma \text{ do}\)
   3. $q \leftarrow \delta(m, a)$
   4. \(\text{for } b \in \Sigma \text{ do}\)
   5. \(\text{if } q = \bot \text{ then } \lambda(a, b) \leftarrow \bot\)
   6. \(\text{else } \lambda(a, b) \leftarrow \delta(q, b)\)
7. $t[n..n+m-1] \leftarrow p$
8. $j \leftarrow m - 1$
9. \(\text{while } j < n \text{ do}\)
10. $q \leftarrow \lambda(t[j], t[j - 1])$
11. \(\text{while } q = \bot \text{ do}\)
12. $j \leftarrow j + m - 1$
13. $q \leftarrow \lambda(t[j], t[j - 1])$
14. $i \leftarrow j - 2$
15. \(\text{while } q \neq \bot \text{ do}\)
16. $q \leftarrow \delta(q, t[i])$
17. $i \leftarrow i - 1$
18. \(\text{if } i < j - m + 1 \text{ then}\)
19. \(\text{output}(j)\)
20. $i \leftarrow i + 1$
21. $j \leftarrow j + i + m$
The idea of looking for the forward character for shifting has been originally introduced by Sunday in the \textbf{Quick-Search} algorithm [5] and then efficiently implemented in the \textbf{Forward-Fast-Search} algorithm [6] and in the \textbf{Shift-And-Sunday} algorithm [7].

\begin{tabular}{c}
\text{t} \\
\text{p}
\end{tabular}
Looking for the forward character

The idea of looking for the forward character for shifting has been originally introduced by Sunday in the **Quick-Search** algorithm [5] and then efficiently implemented in the **Forward-Fast-Search** algorithm [6] and in the **Shift-And-Sunday** algorithm [7].

```
P  t
```

Looking for the forward character

The idea of looking for the forward character for shifting has been originally introduced by Sunday in the **Quick-Search** algorithm [5] and then efficiently implemented in the **Forward-Fast-Search** algorithm [6] and in the **Shift-And-Sunday** algorithm [7].

Looking for the forward character

The idea of looking for the forward character for shifting has been originally introduced by Sunday in the Quick-Search algorithm [5] and then efficiently implemented in the Forward-Fast-Search algorithm [6] and in the Shift-And-Sunday algorithm [7].

---

The Forward-Factor-Oracle

The forward factor oracle of a word $p$, $\text{FOracle}(p)$, is an automaton which recognizes at least all the factors of $p$, eventually preceded by a word $x \in \Sigma \cup \{\varepsilon\}$. More formally the language recognized by $\text{FOracle}(p)$ is defined by

$$\mathcal{L}(\text{FOracle}(p)) = \{xw \mid x \in \Sigma \cup \{\varepsilon\} \text{ and } w \in \mathcal{L}(\text{Oracle}(p))\}$$
Suppose Oracle(p) = \{Q, m, Q_\delta, \Sigma\}, for a pattern p of length m.

\text{FOracle}(p) is an automaton \{Q', (m + 1), Q, \Sigma, \delta'\}, where

1. \( Q' = Q \cup \{(m + 1)\} \)
2. \((m + 1)\) is the initial state
3. all states are final
4. \(\delta'(q, c) = \delta(q, c)\) for all \(c \in \Sigma\), if \(q \neq (m + 1)\)
5. \(\delta'(m + 1, c) = \{m, \delta(m, c)\}\) for all \(c \in \Sigma\)
The Forward-Factor-Oracle

Suppose Oracle(p) = \{Q, m, Q, δ, Σ\}, for a pattern p of length m.

FOracle(p) is an automaton \{Q', (m + 1), Q, Σ, δ'\}, where

1. Q' = Q U \{(m + 1)\}
2. (m + 1) is the initial state
3. all states are final
4. δ'(q, c) = δ(q, c) for all c ∈ Σ, if q ≠ (m + 1)
5. δ'(m + 1, c) = \{m, δ(m, c)\} for all c ∈ Σ
The simulation of the forward factor oracle can be done by simply changing the computation of the table in the following way

\[
\lambda(a, b) = \begin{cases} 
\delta(m, b) & \text{if } \delta(m, a) = \bot \lor b = p[m - 1] \\
\delta(\delta(m, a), b) & \text{otherwise}
\end{cases}
\]
The simulation of the forward factor oracle can be done by simply changing the computation of the table in the following way:

\[ \lambda(a, b) = \begin{cases} 
\delta(m, b) & \text{if } \delta(m, a) = \bot \lor b = p[m - 1] \\
\delta(\delta(m, a), b) & \text{otherwise}
\end{cases} \]
FORWARD-BOM\( (p, m, t, n) \)

1. \( \delta \leftarrow \text{precompute-factor-oracle}(p) \)
2. \textbf{for} \( a \in \Sigma \) \textbf{do}
3. \( q \leftarrow \delta(m, a) \)
4. \textbf{for} \( b \in \Sigma \) \textbf{do}
5. \quad \textbf{if} \( q = \bot \) \textbf{then} \( \lambda(a, b) \leftarrow \bot \)
6. \quad \textbf{else} \( \lambda(a, b) \leftarrow \delta(q, b) \)
7. \( q \leftarrow \delta(m, p[m-1]) \)
8. \textbf{for} \( a \in \Sigma \) \textbf{do} \( \lambda(a, p[m-1]) \leftarrow q \)
9. \( t[n..n+m-1] \leftarrow p \)
10. \( j \leftarrow m - 1 \)
11. \textbf{while} \( j < n \) \textbf{do}
12. \( q \leftarrow \lambda(t[j+1], t[j]) \)
13. \textbf{while} \( q = \bot \) \textbf{do}
14. \quad \( j \leftarrow j + m \)
15. \quad \( q \leftarrow \lambda(t[j+1], t[j]) \)
16. \quad \( i \leftarrow j - 1 \)
17. \textbf{while} \( q \neq \bot \) \textbf{do}
18. \quad \( q \leftarrow \delta(q, t[i]) \)
19. \quad \( i \leftarrow i - 1 \)
20. \textbf{if} \( i < j - m + 1 \) \textbf{then}
21. \quad \text{output}(j)
22. \quad \( i \leftarrow 1 + 1 \)
23. \( j \leftarrow j + i + m \)
FORWARD-BOM\((p, m, t, n)\)

1. \(\delta \leftarrow \text{precompute-factor-oracle}(p)\)
2. \(\textbf{for } a \in \Sigma \textbf{ do}\)
3. \(q \leftarrow \delta(m, a)\)
4. \(\textbf{for } b \in \Sigma \textbf{ do}\)
5. \(\textbf{if } q = \bot \textbf{ then } \lambda(a, b) \leftarrow \bot\)
6. \(\textbf{else } \lambda(a, b) \leftarrow \delta(q, b)\)
7. \(q \leftarrow \delta(m, p[m - 1])\)
8. \(\textbf{for } a \in \Sigma \textbf{ do } \lambda(a, p[m - 1]) \leftarrow q\)
9. \(t[n..n + m - 1] \leftarrow p\)
10. \(j \leftarrow m - 1\)
11. \(\textbf{while } j < n \textbf{ do}\)
12. \(q \leftarrow \lambda(t[j + 1], t[j])\)
13. \(\textbf{while } q = \bot \textbf{ do}\)
14. \(j \leftarrow j + m\)
15. \(q \leftarrow \lambda(t[j + 1], t[j])\)
16. \(i \leftarrow j - 1\)
17. \(\textbf{while } q \neq \bot \textbf{ do}\)
18. \(q \leftarrow \delta(q, t[i])\)
19. \(i \leftarrow i - 1\)
20. \(\textbf{if } i < j - m + 1 \textbf{ then}\)
21. \(\text{output}(j)\)
22. \(i \leftarrow 1 + 1\)
23. \(j \leftarrow j + i + m\)
Efficient Variants of the BOM Algorithm – Simone Faro and Thierry Lecroq – PSC 2008

\[
\text{FORWARD-BOM}(p, m, t, n)
\]

1. \( \delta \leftarrow \text{precompute-factor-oracle}(p) \)
2. \( \text{for } a \in \Sigma \text{ do} \)
3. \( q \leftarrow \delta(m, a) \)
4. \( \text{for } b \in \Sigma \text{ do} \)
5. \( \text{if } q = \bot \text{ then } \lambda(a, b) \leftarrow \bot \)
6. \( \text{else } \lambda(a, b) \leftarrow \delta(q, b) \)
7. \( q \leftarrow \delta(m, p[m-1]) \)
8. \( \text{for } a \in \Sigma \text{ do } \lambda(a, p[m-1]) \leftarrow q \)
9. \( t[n \ldots n+m-1] \leftarrow p \)
10. \( j \leftarrow m-1 \)
11. \( \text{while } j < n \text{ do} \)
12. \( q \leftarrow \lambda(t[j+1], t[j]) \)
13. \( \text{while } q = \bot \text{ do} \)
14. \( j \leftarrow j + m \)
15. \( q \leftarrow \lambda(t[j+1], t[j]) \)
16. \( i \leftarrow j - 1 \)
17. \( \text{while } q \neq \bot \text{ do} \)
18. \( q \leftarrow \delta(q, t[i]) \)
19. \( i \leftarrow i - 1 \)
20. \( \text{if } i < j - m + 1 \text{ then} \)
21. \( \text{output}(j) \)
22. \( i \leftarrow i + 1 \)
23. \( j \leftarrow j + i + m \)
Experimental Results

- Extended-BOM algorithm (EBOM)
- Forward-BOM algorithm (FBOM)
- Forward-SBNDM algorithm (FSBNDM)
- Fast-Search algorithm (FS)
- Forward-Fast-Search algorithm (FFS)
- BOM algorithm (BOM)
- q-Hash algorithms with q=3,5,8 (3-HASH, 5-HASH, 8-HASH)
- SBNDM algorithm (SBNDM)
Experimental Results

- All algorithms have been implemented in the C programming language and were used to search for the same strings in large fixed text buffers.

- The algorithms have been tested:
  - on seven Randσ problems, for σ = 2, 4, 8, 16, 32, 64;
  - on a genome sequence (Escherichia Coli);
  - on a protein sequence (from human genome);

- Searching have been performed for patterns of length m = 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024.

- In the following tables, running times are expressed in hundredths of seconds.
# Experimental Results

<table>
<thead>
<tr>
<th>m</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>153.52</td>
<td>129.07</td>
<td>209.07</td>
<td>169.22</td>
<td>177.31</td>
<td>162.98</td>
<td>-</td>
<td>-</td>
<td>155.38</td>
<td>145.47</td>
</tr>
<tr>
<td>8</td>
<td>115.44</td>
<td>94.42</td>
<td>133.73</td>
<td>105.08</td>
<td>114.17</td>
<td>77.03</td>
<td>67.91</td>
<td>-</td>
<td>87.42</td>
<td>80.72</td>
</tr>
<tr>
<td>16</td>
<td>83.60</td>
<td>63.05</td>
<td>71.75</td>
<td>58.91</td>
<td>63.23</td>
<td>51.65</td>
<td>25.27</td>
<td>34.33</td>
<td>44.87</td>
<td>41.31</td>
</tr>
<tr>
<td>32</td>
<td>61.96</td>
<td>43.40</td>
<td>38.55</td>
<td>30.58</td>
<td>33.24</td>
<td>45.38</td>
<td>14.85</td>
<td>13.50</td>
<td>23.88</td>
<td>20.77</td>
</tr>
<tr>
<td>64</td>
<td>48.16</td>
<td>32.69</td>
<td>21.24</td>
<td>17.43</td>
<td>17.91</td>
<td>44.65</td>
<td>11.53</td>
<td>7.42</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>39.55</td>
<td>24.90</td>
<td>11.91</td>
<td>11.73</td>
<td>15.63</td>
<td>44.02</td>
<td>10.09</td>
<td>8.34</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>32.80</td>
<td>21.14</td>
<td>8.45</td>
<td>8.43</td>
<td>10.00</td>
<td>44.92</td>
<td>11.02</td>
<td>6.86</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>28.07</td>
<td>17.27</td>
<td>6.36</td>
<td>4.87</td>
<td>5.87</td>
<td>45.65</td>
<td>10.04</td>
<td>6.21</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>23.39</td>
<td>15.47</td>
<td>4.00</td>
<td>2.79</td>
<td>3.95</td>
<td>44.72</td>
<td>10.59</td>
<td>5.14</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a Rand2 problem

---

<table>
<thead>
<tr>
<th>m</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>82.12</td>
<td>78.03</td>
<td>111.55</td>
<td>58.93</td>
<td>84.93</td>
<td>117.82</td>
<td>-</td>
<td>-</td>
<td>60.57</td>
<td>74.73</td>
</tr>
<tr>
<td>8</td>
<td>60.00</td>
<td>54.02</td>
<td>61.31</td>
<td>43.57</td>
<td>51.38</td>
<td>43.77</td>
<td>56.23</td>
<td>-</td>
<td>40.46</td>
<td>40.79</td>
</tr>
<tr>
<td>16</td>
<td>49.05</td>
<td>39.49</td>
<td>35.58</td>
<td>29.11</td>
<td>31.66</td>
<td>22.40</td>
<td>20.54</td>
<td>33.98</td>
<td>23.49</td>
<td>23.15</td>
</tr>
<tr>
<td>32</td>
<td>41.72</td>
<td>30.56</td>
<td>19.98</td>
<td>16.88</td>
<td>18.13</td>
<td>16.27</td>
<td>10.60</td>
<td>12.70</td>
<td>12.97</td>
<td>12.48</td>
</tr>
<tr>
<td>64</td>
<td>37.11</td>
<td>23.71</td>
<td>11.63</td>
<td>9.79</td>
<td>11.11</td>
<td>13.53</td>
<td>7.05</td>
<td>7.11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>32.02</td>
<td>18.43</td>
<td>8.30</td>
<td>7.56</td>
<td>10.20</td>
<td>12.17</td>
<td>7.05</td>
<td>8.12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>28.54</td>
<td>15.72</td>
<td>6.27</td>
<td>5.72</td>
<td>6.16</td>
<td>12.25</td>
<td>6.97</td>
<td>6.99</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>26.07</td>
<td>14.13</td>
<td>3.52</td>
<td>3.31</td>
<td>3.67</td>
<td>12.10</td>
<td>7.46</td>
<td>5.71</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>22.14</td>
<td>12.97</td>
<td>1.83</td>
<td>2.25</td>
<td>2.78</td>
<td>11.46</td>
<td>8.02</td>
<td>4.76</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a Rand4 problem
## Experimental Results

<table>
<thead>
<tr>
<th>$m$</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>46.34</td>
<td>44.32</td>
<td>78.94</td>
<td>26.81</td>
<td>49.87</td>
<td>105.12</td>
<td>-</td>
<td>-</td>
<td>33.82</td>
<td>40.05</td>
</tr>
<tr>
<td>8</td>
<td>29.61</td>
<td>27.46</td>
<td>43.23</td>
<td>16.85</td>
<td>30.11</td>
<td>37.30</td>
<td>54.47</td>
<td>-</td>
<td>18.70</td>
<td>23.09</td>
</tr>
<tr>
<td>64</td>
<td>18.93</td>
<td>14.14</td>
<td>8.70</td>
<td>7.07</td>
<td>7.95</td>
<td>8.71</td>
<td>7.87</td>
<td>7.09</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>17.85</td>
<td>12.10</td>
<td>6.99</td>
<td>6.66</td>
<td>7.85</td>
<td>7.11</td>
<td>7.81</td>
<td>7.98</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>17.15</td>
<td>11.13</td>
<td>5.26</td>
<td>3.56</td>
<td>4.70</td>
<td>7.68</td>
<td>6.48</td>
<td>7.43</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>16.02</td>
<td>11.29</td>
<td>3.25</td>
<td>2.61</td>
<td>2.38</td>
<td>7.75</td>
<td>6.53</td>
<td>6.03</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>15.35</td>
<td>9.63</td>
<td>1.88</td>
<td>1.55</td>
<td>1.61</td>
<td>6.91</td>
<td>6.56</td>
<td>5.57</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a Rand8 problem

<table>
<thead>
<tr>
<th>$m$</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>33.17</td>
<td>32.13</td>
<td>52.02</td>
<td>20.09</td>
<td>39.26</td>
<td>102.31</td>
<td>-</td>
<td>-</td>
<td>28.16</td>
<td>27.98</td>
</tr>
<tr>
<td>8</td>
<td>18.52</td>
<td>18.91</td>
<td>35.48</td>
<td>10.73</td>
<td>21.87</td>
<td>34.74</td>
<td>54.09</td>
<td>-</td>
<td>14.04</td>
<td>15.22</td>
</tr>
<tr>
<td>16</td>
<td>13.48</td>
<td>13.01</td>
<td>19.61</td>
<td>6.98</td>
<td>13.76</td>
<td>16.33</td>
<td>18.71</td>
<td>33.78</td>
<td>7.66</td>
<td>9.18</td>
</tr>
<tr>
<td>32</td>
<td>11.41</td>
<td>10.83</td>
<td>9.33</td>
<td>6.36</td>
<td>8.29</td>
<td>9.46</td>
<td>8.64</td>
<td>13.35</td>
<td>6.80</td>
<td>6.43</td>
</tr>
<tr>
<td>64</td>
<td>10.54</td>
<td>9.57</td>
<td>6.74</td>
<td>5.58</td>
<td>7.12</td>
<td>6.79</td>
<td>6.21</td>
<td>7.29</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>10.39</td>
<td>9.14</td>
<td>7.58</td>
<td>5.05</td>
<td>9.99</td>
<td>6.25</td>
<td>8.52</td>
<td>7.03</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>9.88</td>
<td>9.08</td>
<td>5.00</td>
<td>3.16</td>
<td>4.45</td>
<td>6.84</td>
<td>6.98</td>
<td>7.07</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>10.23</td>
<td>9.10</td>
<td>2.55</td>
<td>2.18</td>
<td>2.61</td>
<td>6.22</td>
<td>5.90</td>
<td>6.44</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>10.14</td>
<td>8.55</td>
<td>1.57</td>
<td>1.18</td>
<td>1.45</td>
<td>6.33</td>
<td>5.40</td>
<td>5.62</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a Rand16 problem
## Experimental Results

<table>
<thead>
<tr>
<th>$m$</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>28.04</td>
<td>26.91</td>
<td>35.98</td>
<td>19.03</td>
<td>35.98</td>
<td>100.51</td>
<td>-</td>
<td>-</td>
<td>26.88</td>
<td>23.75</td>
</tr>
<tr>
<td>8</td>
<td>15.51</td>
<td>15.23</td>
<td>24.54</td>
<td>8.98</td>
<td>20.74</td>
<td>34.34</td>
<td>53.71</td>
<td>-</td>
<td>12.28</td>
<td>12.54</td>
</tr>
<tr>
<td>16</td>
<td>9.78</td>
<td>9.44</td>
<td>17.46</td>
<td>6.18</td>
<td>11.56</td>
<td>15.44</td>
<td>18.36</td>
<td>34.14</td>
<td>6.95</td>
<td>7.46</td>
</tr>
<tr>
<td>32</td>
<td>8.29</td>
<td>7.98</td>
<td>10.26</td>
<td>5.46</td>
<td>7.11</td>
<td>8.36</td>
<td>9.02</td>
<td>13.16</td>
<td>5.59</td>
<td>5.75</td>
</tr>
<tr>
<td>64</td>
<td>7.50</td>
<td>7.35</td>
<td>5.78</td>
<td>5.58</td>
<td>6.37</td>
<td>6.37</td>
<td>6.22</td>
<td>7.07</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>7.38</td>
<td>7.70</td>
<td>6.21</td>
<td>3.36</td>
<td>10.62</td>
<td>7.58</td>
<td>8.21</td>
<td>8.32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>7.59</td>
<td>8.33</td>
<td>3.62</td>
<td>2.38</td>
<td>5.94</td>
<td>6.73</td>
<td>6.95</td>
<td>6.75</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>7.89</td>
<td>8.91</td>
<td>1.96</td>
<td>1.41</td>
<td>3.28</td>
<td>6.28</td>
<td>5.78</td>
<td>6.40</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>7.84</td>
<td>7.73</td>
<td>1.57</td>
<td>1.45</td>
<td>1.39</td>
<td>5.91</td>
<td>5.31</td>
<td>5.83</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a Rand32 problem

<table>
<thead>
<tr>
<th>$m$</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>23.55</td>
<td>27.38</td>
<td>29.10</td>
<td>18.79</td>
<td>35.38</td>
<td>97.23</td>
<td>-</td>
<td>-</td>
<td>25.05</td>
<td>23.67</td>
</tr>
<tr>
<td>8</td>
<td>13.48</td>
<td>13.82</td>
<td>18.51</td>
<td>8.76</td>
<td>19.41</td>
<td>33.79</td>
<td>53.80</td>
<td>-</td>
<td>12.15</td>
<td>11.37</td>
</tr>
<tr>
<td>16</td>
<td>8.06</td>
<td>8.44</td>
<td>12.64</td>
<td>5.69</td>
<td>11.35</td>
<td>15.07</td>
<td>18.56</td>
<td>33.32</td>
<td>6.72</td>
<td>6.72</td>
</tr>
<tr>
<td>32</td>
<td>7.04</td>
<td>6.47</td>
<td>9.33</td>
<td>5.14</td>
<td>7.20</td>
<td>8.09</td>
<td>9.00</td>
<td>13.15</td>
<td>5.55</td>
<td>5.25</td>
</tr>
<tr>
<td>64</td>
<td>6.44</td>
<td>6.68</td>
<td>6.34</td>
<td>5.16</td>
<td>6.52</td>
<td>6.13</td>
<td>6.09</td>
<td>7.23</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>8.41</td>
<td>8.24</td>
<td>6.05</td>
<td>3.84</td>
<td>9.85</td>
<td>8.51</td>
<td>7.72</td>
<td>8.45</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>8.82</td>
<td>8.49</td>
<td>3.19</td>
<td>1.96</td>
<td>5.59</td>
<td>7.08</td>
<td>6.52</td>
<td>7.21</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>8.52</td>
<td>9.14</td>
<td>1.99</td>
<td>1.28</td>
<td>3.21</td>
<td>6.05</td>
<td>5.79</td>
<td>6.07</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>8.60</td>
<td>8.36</td>
<td>2.41</td>
<td>1.33</td>
<td>1.64</td>
<td>6.25</td>
<td>4.10</td>
<td>5.67</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a Rand64 problem
# Experimental Results

<table>
<thead>
<tr>
<th>$m$</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>18.64</td>
<td>16.91</td>
<td>23.25</td>
<td>12.65</td>
<td>19.09</td>
<td>25.48</td>
<td>-</td>
<td>-</td>
<td>12.96</td>
<td>17.30</td>
</tr>
<tr>
<td>16</td>
<td>11.48</td>
<td>8.47</td>
<td>7.73</td>
<td>6.77</td>
<td>6.47</td>
<td>4.76</td>
<td>4.39</td>
<td>7.74</td>
<td>5.28</td>
<td>5.50</td>
</tr>
<tr>
<td>32</td>
<td>9.58</td>
<td>6.44</td>
<td>4.53</td>
<td>3.52</td>
<td>4.07</td>
<td>3.20</td>
<td>2.77</td>
<td>2.85</td>
<td>3.04</td>
<td>2.62</td>
</tr>
<tr>
<td>64</td>
<td>8.56</td>
<td>4.92</td>
<td>2.50</td>
<td>1.95</td>
<td>2.42</td>
<td>2.65</td>
<td>1.60</td>
<td>1.84</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>7.05</td>
<td>4.01</td>
<td>1.74</td>
<td>1.73</td>
<td>1.91</td>
<td>2.42</td>
<td>1.84</td>
<td>2.08</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>6.41</td>
<td>3.35</td>
<td>1.33</td>
<td>1.32</td>
<td>1.33</td>
<td>2.90</td>
<td>1.60</td>
<td>1.41</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>5.66</td>
<td>3.20</td>
<td>0.94</td>
<td>0.82</td>
<td>0.78</td>
<td>2.39</td>
<td>1.60</td>
<td>1.61</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>5.97</td>
<td>2.19</td>
<td>0.98</td>
<td>0.66</td>
<td>0.51</td>
<td>2.50</td>
<td>1.21</td>
<td>1.21</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a genome sequence ($\sigma = 4$)

<table>
<thead>
<tr>
<th>$m$</th>
<th>FS</th>
<th>FFS</th>
<th>BOM</th>
<th>EBOM</th>
<th>FBOM</th>
<th>3-HASH</th>
<th>5-HASH</th>
<th>8-HASH</th>
<th>SBNDM</th>
<th>FSBNDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4.33</td>
<td>2.93</td>
<td>8.30</td>
<td>2.14</td>
<td>5.51</td>
<td>14.40</td>
<td>-</td>
<td>-</td>
<td>5.19</td>
<td>3.59</td>
</tr>
<tr>
<td>8</td>
<td>1.68</td>
<td>2.64</td>
<td>4.21</td>
<td>2.27</td>
<td>3.58</td>
<td>4.38</td>
<td>8.09</td>
<td>-</td>
<td>2.31</td>
<td>1.85</td>
</tr>
<tr>
<td>16</td>
<td>1.71</td>
<td>1.57</td>
<td>2.66</td>
<td>1.05</td>
<td>1.92</td>
<td>2.50</td>
<td>2.58</td>
<td>4.54</td>
<td>1.25</td>
<td>1.05</td>
</tr>
<tr>
<td>32</td>
<td>1.41</td>
<td>1.47</td>
<td>1.62</td>
<td>0.87</td>
<td>1.27</td>
<td>1.30</td>
<td>1.37</td>
<td>1.64</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td>64</td>
<td>1.21</td>
<td>1.02</td>
<td>1.10</td>
<td>0.63</td>
<td>1.18</td>
<td>0.85</td>
<td>0.82</td>
<td>1.25</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>1.09</td>
<td>1.33</td>
<td>1.13</td>
<td>0.67</td>
<td>1.51</td>
<td>0.98</td>
<td>1.14</td>
<td>1.22</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>256</td>
<td>1.37</td>
<td>1.44</td>
<td>0.59</td>
<td>0.51</td>
<td>0.47</td>
<td>0.90</td>
<td>0.90</td>
<td>0.82</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>1.20</td>
<td>1.56</td>
<td>0.50</td>
<td>0.27</td>
<td>0.30</td>
<td>0.77</td>
<td>0.90</td>
<td>0.88</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>1.25</td>
<td>1.64</td>
<td>0.39</td>
<td>0.35</td>
<td>0.27</td>
<td>0.87</td>
<td>0.70</td>
<td>0.74</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Running times for a protein sequence ($\sigma = 22$)
Conclusions

We presented two efficient variants of the Backward Oracle Matching algorithm which is considered one of the most effective algorithm for exact string matching.

The first variation, called Extended-BOM, introduces an efficient fast-loop over transitions of the oracle by reading two consecutive characters for each iteration.

The second variation, called Forward-BOM, extends the previous one by using a look-ahead character at the beginning of transitions in order to obtain larger shift advancements.

It turns out from experimental results that the new proposed variations are very fast in practice and obtain the best results in most cases, especially for long patterns and alphabets of medium dimension.