Swap Matching in Strings by Simulating Reactive Automata

Simone Faro

Department of Mathematics and Computer Science
Department of Linguistics and Humanities
University of Catania (Italy)

web-page: http://www.dmi.unict.it/~faro
email: faro@dmi.unict.it

Prague Stringology Conference
September 2, 2013
The Swap Matching Problem

Definition

The string matching problem with swaps is a well-studied variant of the classic string matching problem. It consists in finding all occurrences of a pattern $P$ of length $m$ in a text $T$ of length $n$ up to character swaps.

Constraints

- Each character can be involved in at most one swap;
- Identical adjacent characters are not allowed to be swapped.

Example

$P : agtgac$
$T : gtagatagccgatatggacacga$
The Swap Matching Problem

Definition

The string matching problem with swaps is a well-studied variant of the classic string matching problem. It consists in finding all occurrences of a pattern $P$ of length $m$ in a text $T$ of length $n$ up to character swaps.

Constraints

- Each character can be involved in at most one swap;
- Identical adjacent characters are not allowed to be swapped.

Example

$P : \text{agtgac}$
$T : \text{gta} \text{gatagc} \text{cgatatggacacga}$
The string matching problem with swaps is a well-studied variant of the classic string matching problem. It consists in finding all occurrences of a pattern $P$ of length $m$ in a text $T$ of length $n$ up to character swaps.

- Each character can be involved in at most one swap;
- Identical adjacent characters are not allowed to be swapped.

**Example**

$P : agtgac$
$T : gtagatagccgat_{\text{atggacacga}}$
The Swap Matching Problem

Previous Theoretical Results

- **1995** The problem was introduced by Muthukrishnan;
- **1997** The first nontrivial result was reported by Amir et al. who provided a $O(n m^{\frac{1}{3}} \log m)$-time algorithm in the case of binary alphabet;
- **1998** Amir et al. obtained a $O(m \log^2 m)$ solution on some restrictive cases;
- **2003** Amir et al. solved the problem in $O(n \log m \log \sigma)$-time.
Iliopoulos and Rahman

2008. The first attempt to provide an efficient solution to the swap matching problem without using the FFT technique has been presented by Iliopoulos and Rahman. They introduced a new graph-theoretic approach to model the problem and devised an efficient algorithm, based on the bit-parallelism technique, which runs in $O((n + m) \log m)$-time, in the case of short patterns.
The Swap Matching Problem

Iliopoulos and Rahman

2008. The first attempt to provide an efficient solution to the swap matching problem without using the FFT technique has been presented by Iliopoulos and Rahman. They introduced a new graph-theoretic approach to model the problem and devised an efficient algorithm, based on the bit-parallelism technique, which runs in $O((n + m) \log m)$-time, in the case of short patterns.

Cantone and Faro

2009. Cantone and Faro presented a new efficient algorithm, named Cross-Sampling (CS), which simulates a non-deterministic automaton with $2m$ states and $3m - 2$ transitions. It admits an efficient bit-parallel implementation, named Bit-Parallel-Cross-Sampling (BPCS), which achieves $O(n)$ worst-case time and $O(\sigma)$ space complexity in the case of short patterns.
The Standard Automaton

Cantone & Faro (2009)

Dimension: 2m states and 3m – 2 transitions
Simulation: 7 bitwise operations
The Standard Automaton

Cantone & Faro (2009)

Dimension: \(2m\) states and \(3m - 2\) transitions
Simulation: 7 bitwise operations

The present work (2013)

Dimension: \(m\) states, \(3m - 2\) transitions (\(8m - 12\) links)
Simulation: 7 operations (or 2 operations under suitable conditions)
Switch Reactive Automata

Definition (Switch Reactive Transformation)

Let $\delta \subseteq (Q \times \Sigma \times Q)$ be the transition relation of an automaton $A$ and let $\varphi \subseteq \delta$. Let $T^+$, $T^-$ be two subsets of $\delta \times \delta$. A transformation $\delta \rightarrow \delta^\varphi$, for $\varphi \subseteq \delta$, is defined as follows

$$
\delta^\varphi = (\delta \setminus \{ \gamma \mid \gamma \in \delta \text{ and } \exists \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^- \}) \\
\cup \{ \gamma \mid \gamma \in \delta \text{ and } \exists \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^+ \}
$$

The reactive links are intended to be applied simultaneously.

![Diagram](diagram.png)
Switch Reactive Automata

Definition (Switch Reactive Transformation)

Let $\delta \subseteq (Q \times \Sigma \times Q)$ be the transition relation of an automaton $A$ and let $\varphi \subseteq \delta$. Let $T^+$, $T^-$ be two subsets of $\delta \times \delta$. A transformation $\delta \rightarrow \delta^\varphi$, for $\varphi \subseteq \delta$, is defined as follows

$$\delta^\varphi = (\delta \setminus \{ \gamma \mid \gamma \in \delta \text{ and } \exists \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^- \}) \cup \{ \gamma \mid \gamma \in \delta \text{ and } \exists \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^+ \}$$

The reactive links are intended to be applied simultaneously.
Definition (Switch Reactive Transformation)

Let $\delta \subseteq (Q \times \Sigma \times Q)$ be the transition relation of an automaton $A$ and let $\varphi \subseteq \delta$. Let $T^+$, $T^-$ be two subsets of $\delta \times \delta$. A transformation $\delta \rightarrow \delta\varphi$, for $\varphi \subseteq \delta$, is defined as follows

$$\delta\varphi = (\delta \setminus \{\gamma \mid \gamma \in \delta \text{ and } \exists \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^-\}) \cup \{\gamma \mid \gamma \in \delta \text{ and } \exists \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^+\}$$

The reactive links are intended to be applied simultaneously.
A reactive automaton is an ordinary non-deterministic automaton with a switch reactive transformation, i.e. a triple $R = (A, T^+, T^-)$ which defines the switch reactive transformation above.
Switch Reactive Automata

Definition (Switch Reactive Automaton)

Reactive automata are used to reduce dramatically the number of states in both deterministic and the non-deterministic automata. A reactive automaton has extra links whose role is to change the behavior of the automaton itself.

A reactive automaton accepting the language $a^{2n+1}$. All arcs are initially active. The automaton uses a reactive edge-to-edge link to cancel or activate the arc from the initial state $i$ to the terminal state $t$. 
Switch Reactive Automata

Definition (Switch Reactive Automaton)

Reactive automata are used to reduce dramatically the number of states in both deterministic and the non-deterministic automata. A reactive automaton has extra links whose role is to change the behavior of the automaton itself.

Two deterministic reactive automata accepting the set of strings in which each letter of the alphabet \( \{a_1, a_2, \ldots, a_k\} \) appears at most once. All loop arcs are initially active. Loops on state \( i \) are made inactive after their first use.
Switch Reactive Automata

**Definition (Switch Reactive Automaton)**

Reactive automata are used to reduce dramatically the number of states in both deterministic and the non-deterministic automata. A reactive automaton has extra links whose role is to change the behavior of the automaton itself.

A deterministic reactive automaton accepting the set of strings that are permutations of the letters $a_1, a_2, \ldots, a_k$. All loops on the initial state are initially active and other $\varepsilon$-arcs are inactive. One reactive link for letter $a_i$ cancels its respective loop while the second activates its associated $\varepsilon$-arc.
Definition (Swap Reactive Automaton)

Let $P$ be a pattern of length $m$ over an alphabet $\Sigma$. The Swap Reactive Automaton (SRA) for $P$ is a Reactive Automaton $R = (A, T^+, T^-)$, with $A = (Q, \Sigma, \delta, q_0, F)$, such that

- $Q = \{q_0, q_1, \ldots, q_m\}$ is the set of states;
- $q_0$ is the initial state;
- $F = \{q_m\}$ is the set of final states;
- $\delta$ is the transition relation;
- $T^+$ is the set of (switch on) reactive links;
- $T^-$ is the set of (switch off) reactive links.
Swap Reactive Automaton

Definition (The Transition Function)

\[ \delta = \{(q_i, p_i, q_{i+1}) \mid 0 \leq i < m\} \cup \{(q_i, p_{i+1}, q_{i+1}) \mid 0 \leq i < m - 1 \text{ and } p_i \neq p_{i+1}\} \cup \{(q_i, p_{i-1}, q_{i+1}) \mid 1 \leq i < m \text{ and } p_i \neq p_{i-1}\} \cup \{(q_0, \Sigma, q_0)\} \]

- no swaps
- start of a swap
- end of a swap
- self loop
Swap Reactive Automaton

Definition (The Transition Function)

\[ \delta = \begin{cases} 
(q_i, p_i, q_{i+1}) & | 0 \leq i < m \} \cup \\
(q_i, p_{i+1}, q_{i+1}) & | 0 \leq i < m - 1 \text{ and } p_i \neq p_{i+1} \} \cup \\
(q_i, p_{i-1}, q_{i+1}) & | 1 \leq i < m \text{ and } p_i \neq p_{i-1} \} \cup \\
(q_0, \Sigma, q_0) \newline
\end{cases} \]

- no swaps
- start of a swap
- end of a swap
- self loop
Swap Reactive Automaton

Definition (The Transition Function)

\[ \delta = \{(q_i, p_i, q_{i+1}) \mid 0 \leq i < m\} \cup \{(q_i, p_{i+1}, q_{i+1}) \mid 0 \leq i < m - 1 \text{ and } p_i \neq p_{i+1}\} \cup \{(q_i, p_{i-1}, q_{i+1}) \mid 1 \leq i < m \text{ and } p_i \neq p_{i-1}\} \cup \{(q_0, \Sigma, q_0)\} \]

- no swaps
- start of a swap
- end of a swap
- self loop
Swap Reactive Automaton

**Definition (The Transition Function)**

\[ \delta = \begin{cases} 
(q_i, p_i, q_{i+1}) & | 0 \leq i < m \} \cup \\
(q_i, p_{i+1}, q_{i+1}) & | 0 \leq i < m - 1 \text{ and } p_i \neq p_{i+1} \} \cup \\
(q_i, p_{i-1}, q_{i+1}) & | 1 \leq i < m \text{ and } p_i \neq p_{i-1} \} \cup \\
(q_0, \Sigma, q_0) \}
\]

- no swaps
- start of a swap
- end of a swap
- self loop
Definition (Switch off reactive links)

\[
T^- = \{ (q_i, p_i, q_{i+1}), (q_{i+1}, p_i, q_{i+2}) \in (\delta \times \delta) \mid 0 \leq i < m - 1 \} \cup \\
\{ (q_i, p_{i-1}, q_{i+1}), (q_{i+1}, p_i, q_{i+2}) \in (\delta \times \delta) \mid 1 \leq i < m - 1 \} \cup \\
\{ (q_i, p_{i+1}, q_i), (q_{i+1}, p_{i+1}, q_{i+2}) \in (\delta \times \delta) \mid 0 \leq i < m - 1 \} \cup \\
\{ (q_i, p_{i+1}, q_{i+1}), (q_{i+1}, p_{i+2}, q_{i+2}) \in (\delta \times \delta) \mid 0 \leq i < m - 2 \}
\]
Swap Reactive Automaton

**Definition (Switch on reactive links)**

\[
T^+ = \left\{ ((q_i, p_i, q_{i+1}), (q_i, p_{i-1}, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m - 1 \right\} \cup \\
\left\{ ((q_i, p_{i+1}, q_{i+1}), (q_i, p_{i-1}, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m - 1 \right\} \cup \\
\left\{ ((q_i, p_{i-1}, q_{i+1}), (q_i, p_i, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m - 1 \right\} \cup \\
\left\{ ((q_i, p_{i-1}, q_{i+1}), (q_i, p_{i+1}, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m - 1 \right\} \cup \\
\left\{ ((q_i, p_{i-1}, q_{i+1}), (q_i, p_{i+1}, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m - 1 \right\}
\]
Introduction

Swap Reactive Automata

Bit Parallel Simulations

Reactive Automata
The Swap Reactive Automata
Working Principles

Swap Reactive Automaton

Working Principles: No Swap

$q_{i-1} \rightarrow q_i \rightarrow q_{i+1} \rightarrow q_{i+2}$

$p_{i-2} \rightarrow p_{i-1} \rightarrow p_i \rightarrow p_{i+1} \rightarrow p_{i+2}$

Simone Faro

Swap Matching in Strings by Simulating Reactive Automata
Swap Reactive Automata

Working Principles: No Swap

\[ q_{i-1} \rightarrow q_i \rightarrow q_{i+1} \rightarrow q_{i+2} \]

- \( p_{i-2} \rightarrow q_{i-1} \)
- \( p_{i-1} \rightarrow q_i \)
- \( p_i \rightarrow q_{i+1} \)
- \( p_{i+1} \rightarrow q_{i+2} \)
- \( p_{i+2} \rightarrow \)
Swap Reactive Automaton

Working Principles: No Swap

$q_{i-1}$ → $q_i$ → $q_{i+1}$ → $q_{i+2}$

$p_{i-1}$ → $p_i$ → $p_{i+1}$ → $p_{i+2}$

$p_{i-2}$ → $p_{i-1}$ → $p_i$ → $p_{i+1}$ → $p_{i+2}$
Swap Reactive Automaton

Working Principles: No Swap

\[ q_{i-1} \xrightarrow{p_{i-2}} q_i \xrightarrow{p_i} p_{i+1} \xrightarrow{p_{i+2}} q_{i+2} \]
Introduction
Swap Reactive Automata
Bit Parallel Simulations

Swapping Reactive Automata

Working Principles

Swap Matching in Strings by Simulating Reactive Automata
Swap Reactive Automaton

Working Principles: Swap

$q_{i-1}$ \(\rightarrow\) \(p_{i-1}\) \(\rightarrow\) \(p_i\) \(\rightarrow\) \(q_i\) \(\rightarrow\) \(q_{i+1}\) \(\rightarrow\) \(p_{i+1}\) \(\rightarrow\) \(p_i\) \(\rightarrow\) \(p_{i-1}\) \(\rightarrow\) \(q_{i+2}\)
Swap Reactive Automaton

Working Principles: Swap

\[ q_{i-1} \rightarrow p_{i-2} \rightarrow q_i \rightarrow p_i \rightarrow q_{i+1} \rightarrow p_{i+1} \rightarrow q_{i+2} \]
Swap Reactive Automaton

Working Principles: Swap
Swap Reactive Automaton

Working Principles: Swap

\[ q_{i-1} \rightarrow p_{i-2} \rightarrow p_{i-1} \rightarrow q_i \rightarrow p_i \rightarrow p_{i+1} \rightarrow q_{i+1} \rightarrow p_{i+2} \rightarrow q_{i+2} \]
Swap Reactive Automaton

Working Principles: Swap

$q_{i-1} \rightarrow q_i \rightarrow q_{i+1} \rightarrow q_{i+2}$

$p_i \rightarrow p_{i+1} \rightarrow p_{i+2}$

$q_{i-1} \rightarrow p_{i-2} \rightarrow p_{i-1} \rightarrow q_i \rightarrow p_i \rightarrow p_{i+1} \rightarrow q_{i+1} \rightarrow p_{i+2} \rightarrow q_{i+2}$
Swap Reactive Automaton

Example

\[ P : \text{agcat} \]
\[ T : \text{gaacgtagact} \]
Swap Reactive Automaton

Example

\[ P : \text{agcat} \]
\[ T : \text{gaacgtagact} \]
Swap Reactive Automaton

Example

P : agcat
T : gaaacgtagact
Swap Reactive Automaton

Example

\[ P : \text{agcat} \]
\[ T : g\text{aacgtagact} \]
Swap Reactive Automaton

Example

P : agcat
T : gaacgtagact

\[ \Sigma \]

\[ \begin{array}{c}
q_0 \\
q_1 \\
q_2 \\
q_3 \\
q_4 \\
q_5 \\
\end{array} \begin{array}{c}
a \\
g \\
c \\
a \\
t \\
\end{array} \begin{array}{c}
a \\
g \\
c \\
a \\
t \\
\end{array} \begin{array}{c}
a \\
g \\
c \\
a \\
t \\
\end{array} \begin{array}{c}
a \\
g \\
c \\
a \\
t \\
\end{array} \begin{array}{c}
q_0 \\
q_1 \\
q_2 \\
q_3 \\
q_4 \\
q_5 \\
\end{array} \]
Swap Reactive Automaton

Example

\[ P : \text{agcat} \]
\[ T : \text{gaa\textcolor{red}{c}gtagact} \]
**Introduction**

**Swap Reactive Automata**

**Bit Parallel Simulations**

---

**Swapping Reactive Automaton**

**Example**

P : agcat  
T : gaacgtagact

---

Simone Faro

Swap Matching in Strings by Simulating Reactive Automata
Swap Reactive Automaton

Example

P : agcat
T : gaacgtagact
Swap Reactive Automaton

Example

P : agcat
T : gaacgtagact
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to \( w \), where \( w \) is the number of bits in the computer word.
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

**Bitwise Operations**

<table>
<thead>
<tr>
<th>Bitwise AND</th>
<th>Bitwise OR</th>
<th>Bitwise SHIFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10011010 &amp;$</td>
<td>$10011010</td>
<td>$10011010 \gg 1$</td>
</tr>
<tr>
<td>$01011001 =$</td>
<td>$01011001 =$</td>
<td>$010011000$</td>
</tr>
<tr>
<td>$00011000$</td>
<td>$11011011$</td>
<td>$01001101$</td>
</tr>
</tbody>
</table>
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

How to simulate a transition

\[
\begin{align*}
\sum &\quad \rightarrow \quad a \quad b \quad a \quad a \quad b \\
0 &\quad \rightarrow \quad 1 \quad 2 \quad 3 \quad 4 \quad 5
\end{align*}
\]
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

**How to simulate a transition**

Transition on $b$

$D$

$\Sigma$

0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5

1 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 0
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

### How to simulate a transition

**Transition on** $b$  
$D = D \gg 1$
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

How to simulate a transition

Transition on $b$
$D = D \gg 1$
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to \( w \), where \( w \) is the number of bits in the computer word.

### How to simulate a transition

Transition on \( b \)

- \( D = D \gg 1 \)
- \( D = D \mid 10^{m-1} \)
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

How to simulate a transition

Transition on $b$

$D = D \gg 1$

$D = D \mid 10^{m-1}$
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

**How to simulate a transition**

Transition on $b$

$D = D\gg 1$

$D = D \mid 10^{m-1}$

$D = D \& B[b]$
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to $w$, where $w$ is the number of bits in the computer word.

How to simulate a transition

Transition on $b$

$D = D \gg 1$

$D = D \mid 10^{m-1}$

$D = D \& B[b]$
The Bit-Parallel Encoding

Encoding the Reactive Automaton

\[ \Sigma \]

\[
\begin{array}{c}
0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \\
\end{array}
\]

\[
\begin{array}{c}
B(b) \\
B(c) \\
B(a)
\end{array}
\]

\[
\begin{array}{c}
0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \\
0 \quad 0 \quad 0 \quad 1 \quad 0 \\
1 \quad 0 \quad 1 \quad 1 \quad 0
\end{array}
\]
The Bit-Parallel Encoding

Encoding the Reactive Automaton

\[ \Sigma \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \]

\[ a \rightarrow b \quad a \rightarrow c \quad b \rightarrow a \quad b \rightarrow c \quad c \rightarrow b \]
The Bit-Parallel Encoding

Encoding the Reactive Automaton

\[ \sum \]

\[
\begin{array}{c}
\text{0} \\
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{5}
\end{array}
\]

\[
\begin{array}{c}
a \\
b \\
a \\
b \\
c \\
b
\end{array}
\]

\[
\begin{array}{c}
a \\
b \\
a \\
c \\
b \\
c
\end{array}
\]

\[
\begin{array}{c}
\Sigma \\
A \\
1 \\
1 \\
0 \\
1 \\
0
\end{array}
\]
The Bit-Parallel Encoding

Encoding the Reactive Automaton

\[ \Sigma \]

\[ \begin{array}{c}
  0 & 1 & 2 & 3 & 4 & 5 \\
  a & b & a & c & b & a \\
  b & a & b & c & b & c \\
  \end{array} \]

\[ \begin{array}{c}
  A \\
  1 & 1 & 0 & 1 & 0 \\
  B \\
  1 & 1 & 0 & 1 & 0 \\
  \end{array} \]
The Bit-Parallel Encoding

Encoding the Reactive Automaton

0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5

| A | 1 | 1 | 0 | 1 | 0 |
| B | 1 | 1 | 0 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on $b$

\[ \begin{align*}
A & : 0 & 0 & 0 & 1 & 0 \\
B & : 1 & 0 & 0 & 0 & 0 \\
C & : 0 & 1 & 0 & 0 & 0 \\
\end{align*} \]
The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on $b$
Computing Vector $A'$

$A' = C \gg 1$
$A' = A' \& M[t_{j-1}]$

![Diagram of the reactive automaton with states 0 to 5 and transitions on symbols $a$, $b$, and $c$.]
The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on $b$
Computing Vector $B'$

$B' = A \gg 1$
$B' = B' \mid (B \gg 1)$
$B' = B' \mid 10^{m-1}$
$B' = B' \& M[t_j]$
The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on $b$
Computing Vector $C'$

$C' = A \gg 1$
$C' = C' \mid (B \gg 1)$
$C' = C' \mid 10^{m-1}$
$C' = C' \& M[t_{j+1}]$
Encoding the Reactive Automaton

Transition on $b$

\[ \begin{align*}
A & : 0 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0 \\
B & : 1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \\
C & : 0 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \\
A' & : 0 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 0 \\
B' & : 0 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 1 \\
C' & : 1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0
\end{align*} \]
The Bit-Parallel Encoding

The Bit-Parallel Swap Reactive Automaton Matcher

\[ \text{BPSRA}(P, m, T, n) \]

1. for \( c \in \Sigma \) do
2. \( M[c] \leftarrow 0 \)
3. for \( i \leftarrow 0 \) to \( m - 1 \) do
4. \( M[p_i] \leftarrow M[p_i] \mid (1 \ll i) \)
5. \( F \leftarrow 1 \ll (m - 1) \)
6. \( A \leftarrow 0 \)
7. \( B \leftarrow 0^{m-1}1 \& M[t_0] \)
8. \( C \leftarrow 0^{m-1}1 \& M[t_1] \)
9. for \( i \leftarrow 1 \) to \( n - 1 \) do
10. \( H \leftarrow (A \ll 1) \mid (M \ll 1) \mid 1 \)
11. \( A \leftarrow (C \ll 1) \& M[t_j] \)
12. \( B \leftarrow H \& M[t_j] \)
13. \( C \leftarrow H \& M[t_{j+1}] \)
14. if \( ((A \mid B) \& F) \) then
15. \( \text{output}(i - m + 1) \)
Definition (String With Disjoint Triplets)

A string $S = s_0 s_1 s_2 \ldots s_{m-1}$, of length $m$, over an alphabet $\Sigma$, is a string with disjoint triplets (SDT) if $s_i \neq s_{i+2}$, for $i = 0, \ldots, m - 3$.

The above definition implies that in the SRA of $S$ the standard transitions from state $q_i$ to $q_{i+1}$, for $i = 0, \ldots, m - 1$, are labeled by different characters.
A More Efficient Simulation

Relative Frequency of SDT in different text buffers

<table>
<thead>
<tr>
<th>Text</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genome Sequence</td>
<td>0.6080</td>
<td>0.2140</td>
<td>0.0170</td>
<td>0.0010</td>
</tr>
<tr>
<td>Protein Sequence</td>
<td>0.8420</td>
<td>0.6160</td>
<td>0.3140</td>
<td>0.1170</td>
</tr>
<tr>
<td>English Text</td>
<td>0.9380</td>
<td>0.8440</td>
<td>0.6820</td>
<td>0.4380</td>
</tr>
<tr>
<td>Italian Text</td>
<td>0.9130</td>
<td>0.7630</td>
<td>0.5100</td>
<td>0.2500</td>
</tr>
<tr>
<td>French Text</td>
<td>0.9230</td>
<td>0.7910</td>
<td>0.5930</td>
<td>0.3250</td>
</tr>
<tr>
<td>Chinese Text</td>
<td>0.9860</td>
<td>0.9510</td>
<td>0.8990</td>
<td>0.7750</td>
</tr>
</tbody>
</table>

For each text buffer data have been collected by extracting 10,000 random patterns of different length (ranging from 4 to 32) from the text, and computing the corresponding frequency of SDT.
In the new proposed simulation the representation of $R$ uses an array $B$ of $\sigma^2$ bit-vectors, each of size $m$, where the $i$-th bit of $B[c_1, c_2]$ (which we indicate as $B[c_1, c_2]_i$) is defined as

$$
B[c_1, c_2]_i = \begin{cases} 
1 & \text{if } (q_i, c_1, q_i+1), (q_i+1, c_2, q_i+2) \in \delta \text{ and } \not ((q_i, c_1, q_i+1), (q_i+1, c_2, q_i+2)) \in T^- \\
0 & \text{otherwise} 
\end{cases}
$$

for $c_1, c_2 \in \Sigma$, and $0 \leq i < m$.

Roughly speaking, the matrix $M$ encodes the couples of admissible consecutive transitions in $R$. 
A More Efficient Simulation

How to simulate a transition

\[ \Sigma \]

0 \xrightarrow{a} 1 \xrightarrow{g} 2 \xrightarrow{c} 3 \xrightarrow{a} 4 \xrightarrow{t} 5
A More Efficient Simulation

How to simulate a transition

\[ B(a, c) \]

\[ 1 \quad 1 \quad 1 \quad 0 \quad 0 \]
A More Efficient Simulation

How to simulate a transition

\[ \Sigma \]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\downarrow & g & c & a & t & \quad \text{0}
\end{array}
\]

\[
B(a, c) = \begin{bmatrix}
1 \\
1 \\
1 \\
0 \\
0
\end{bmatrix}
\]

\[
B(a, a) = \begin{bmatrix}
1 \\
0 \\
1 \\
0 \\
0
\end{bmatrix}
\]
A More Efficient Simulation

How to simulate a transition

\[
\begin{align*}
\Sigma & \rightarrow a \rightarrow g \rightarrow c \rightarrow a \rightarrow t \\
B(a, c) & \quad \begin{array}{ccccccc}
1 & 1 & 1 & 0 & 0 \\
\end{array} \\
B(a, a) & \quad \begin{array}{ccccccc}
1 & 0 & 1 & 0 & 0 \\
\end{array} \\
B(a, g) & \quad \begin{array}{ccccccc}
1 & 0 & 0 & 0 & 0 \\
\end{array}
\end{align*}
\]
Automaton configurations are then encoded as a bit-vector $D$ of $m$ bits (the initial state does not need to be represented), where the $i$-th bit of $D$ is set if and only if the state $q_{i+1}$ is active.
Automaton configurations are then encoded as a bit-vector $D$ of $m$ bits (the initial state does not need to be represented), where the $i$-th bit of $D$ is set if and only if the state $q_{i+1}$ is active.

When a search starts, the configuration $D$ is initialized to $B[t_0, t_1]$. Then, while the string $T$ is read from left to right, the automaton configuration is updated accordingly for each text character.
Automaton configurations are then encoded as a bit-vector $D$ of $m$ bits (the initial state does not need to be represented), where the $i$-th bit of $D$ is set if and only if the state $q_{i+1}$ is active.

When a search starts, the configuration $D$ is initialized to $B[t_0, t_1]$. Then, while the string $T$ is read from left to right, the automaton configuration is updated accordingly for each text character.

Suppose the last transition has been performed on character $t_{j-1}$, with $0 < j < n - 1$, leading to a configuration vector $D$ of the SRA. Then a transition on character $t_j$ can be implemented by the bitwise operations

$$D(j) = \begin{cases} B[t_0, t_1] & \text{if } j = 1 \\ (D(j-1) \ll 1) \text{ and } B[t_{j-1}, t_j] & \text{otherwise} \end{cases}$$
Automaton configurations are then encoded as a bit-vector $D$ of $m$ bits (the initial state does not need to be represented), where the $i$-th bit of $D$ is set if and only if the state $q_{i+1}$ is active.

When a search starts, the configuration $D$ is initialized to $B[t_0, t_1]$. Then, while the string $T$ is read from left to right, the automaton configuration is updated accordingly for each text character.

Suppose the last transition has been performed on character $t_{j-1}$, with $0 < j < n - 1$, leading to a configuration vector $D$ of the SRA. Then a transition on character $t_j$ can be implemented by the bitwise operations

$$D^{(j)} = \begin{cases} B[t_0, t_1] & \text{if } j = 1 \\ (D^{(j-1)} \ll 1) \text{ and } B[t_{j-1}, t_j] & \text{otherwise} \end{cases}$$

It turns out that, if $P$ is a SDT, then the simulation of the SRA described above works properly, as stated by the following lemma.
The Bit-Parallel Encoding

The Bit-Parallel Swap Reactive Oracle Matcher

\[
\text{BPSRO}(P, m, T, n) \\
1. \quad \text{for } c_1, c_2 \in \Sigma \text{ do } B[c_1, c_2] \leftarrow 0 \\
2. \quad \text{for } i = 1 \text{ to } m - 1 \text{ do} \\
3. \quad \quad B[p_{i-1}, p_i] \leftarrow B[p_{i-1}, p_i] \mid (1 \ll i) \\
4. \quad \quad B[p_i, p_{i-1}] \leftarrow B[p_i, p_{i-1}] \mid (1 \ll i) \\
5. \quad \quad \text{if } (i < m - 1) \text{ then} \ \\
6. \quad \quad \quad B[p_{i-1}, p_{i+1}] \leftarrow B[p_{i-1}, p_{i+1}] \mid (1 \ll i) \\
7. \quad \quad \text{if } (i > 1) \text{ then} \ \\
8. \quad \quad \quad B[p_{i-2}, p_i] \leftarrow B[p_{i-2}, p_i] \mid (1 \ll i) \\
9. \quad \quad \quad \text{if } (i < m - 1) \text{ then} \ \\
10. \quad \quad \quad \quad B[p_{i-2}, p_{i+1}] \leftarrow B[p_{i-2}, p_{i+1}] \mid (1 \ll i) \\
11. \quad F \leftarrow 1 \ll (m - 1), D \leftarrow 0 \\
12. \quad \text{for } i = 1 \text{ to } n - 1 \text{ do} \\
13. \quad \quad D \leftarrow ((D \ll 1) \mid 1) \& B[t_{i-1}, t_i] \\
14. \quad \quad \text{if } (D \& F) \text{ then} \ \\
15. \quad \quad \quad \text{if } (P \text{ is a SDT}) \text{ then output}(i - m + 1) \\
16. \quad \quad \quad \text{else check occurrence at position } (i - m + 1)
\]
### Experimental Results

<table>
<thead>
<tr>
<th>m</th>
<th>(A) genome sequence</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>BPCS</td>
<td>16.0</td>
<td>15.9</td>
<td>15.9</td>
<td>16.0</td>
<td>15.9</td>
<td></td>
</tr>
<tr>
<td>BPSRA</td>
<td><strong>15.4</strong></td>
<td>15.3</td>
<td>15.3</td>
<td>15.2</td>
<td>15.2</td>
<td></td>
</tr>
<tr>
<td>BPSRO</td>
<td>20.4</td>
<td><strong>13.7</strong></td>
<td><strong>11.4</strong></td>
<td><strong>11.2</strong></td>
<td><strong>11.2</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>(B) protein sequence</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>BPCS</td>
<td>15.9</td>
<td>15.9</td>
<td>16.1</td>
<td>16.1</td>
<td>16.2</td>
<td></td>
</tr>
<tr>
<td>BPSRA</td>
<td>15.3</td>
<td>15.8</td>
<td>15.4</td>
<td>15.3</td>
<td>15.3</td>
<td></td>
</tr>
<tr>
<td>BPSRO</td>
<td><strong>12.0</strong></td>
<td><strong>11.2</strong></td>
<td><strong>11.4</strong></td>
<td><strong>11.3</strong></td>
<td><strong>11.3</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>(C) natural language text</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>BPCS</td>
<td>16.0</td>
<td>15.9</td>
<td>16.1</td>
<td>16.3</td>
<td>16.0</td>
<td></td>
</tr>
<tr>
<td>BPSRA</td>
<td>15.3</td>
<td>15.4</td>
<td>15.3</td>
<td>15.3</td>
<td>15.3</td>
<td></td>
</tr>
<tr>
<td>BPSRO</td>
<td><strong>12.8</strong></td>
<td><strong>11.5</strong></td>
<td><strong>11.5</strong></td>
<td><strong>11.3</strong></td>
<td><strong>11.3</strong></td>
<td></td>
</tr>
</tbody>
</table>