Efficient validation and construction of border arrays

J.-P. Duval    T. Lecroq    A. Lefebvre

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes
University of Rouen, France

JM 2006
30/08/2006 Rennes, France
1. Recalls

2. New results

3. Conclusions and perspectives
Outline

1. Recalls
2. New results
3. Conclusions and perspectives
Definition
A string $u$ is a border of a string $w$ if $u$ is both a prefix and a suffix of $w$ such that $u \neq w$.

Definition
The border of a string $w$ is the longest of its borders. It is denoted by $\text{Border}(w)$. 
**Definition**

Given a string $w[1..n]$ of length $n$, the array $f$ defined by

$$f[i] = |Border(w[1..i])|$$

for $1 \leq i \leq n$ is called the **border array** of $w$.

It constitutes the “failure function” of the Morris-Pratt (1970) string matching algorithm.
### Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>12</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w[i]$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$f[i]$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
The DFA $\mathcal{D}(w)$ recognizing the language $\Sigma^*w$ is defined by $\mathcal{D}(w[1..n]) = (Q, \Sigma, q_0, T, F)$ where

- $Q = \{0, 1, \ldots, n\}$ is the set of states;
- $\Sigma$ is the alphabet;
- $q_0 = 0$ is the initial state;
- $T = \{n\}$ is the set of accepting states;
- $F = \{(i, w[i+1], i+1) \mid 1 \leq i \leq n\} \cup \{(i, a, |\text{Border}(w[1..i]a)|) \mid 0 \leq i < n \text{ and } a \in \Sigma \setminus \{w[i+1]\}\}$ is the set of transitions.

The underlying unlabeled graph is called the skeleton of the automaton.
DFA

Example

$D(aabab)$: transitions leading to state 0 are omitted.
Definition

For $0 \leq i \leq n$:

- $\delta(i) = \{ j \mid (i, a, j) \in F \text{ with } a \in A \text{ and } j \neq 0 \}$;
- $\delta'(i) = \{ j \mid (i, a, j) \in F \text{ with } a \in A \text{ and } j \notin \{0, i + 1\} \}$.

In words:

- $\delta(i)$ is the list of the targets of the significant transitions leaving state $i$;
- $\delta'(i)$ is the list of the targets of the backward significant transitions leaving state $i$. 
Example

\[ D(aabab): \text{transitions leading to state 0 are omitted.} \]

\[ \delta(4) = (5, 2) \text{ and } \delta'(4) = (2). \]
Theorem 1 [Simon 1993]

There are at most $n$ significant backward transitions in $D(w[1..n])$. 
Definition

An integer array $f[1..n]$ is a valid array (or is valid) if and only if it is the border array of at least one string $w[1..n]$. 
The main problems

**Validation**
Given an integer array, is it valid? On which alphabet size?

**Construction of a string**
Given a valid array, exhibit a string for which this array is the border array?

**Construction of border arrays**
Construct all the distinct border arrays for some length.
Motivations

Theoretical interest

Generating minimal test sets for various string algorithms
Previous works

Counting distinct strings.

Verifying a border array in linear time.

J.-P. Duval, T. Lecroq, and A. Lefebvre.
Border array on bounded alphabet.
Previous works

Web site

http://al.jalix.org/Baba/Applet/baba.php
The candidates

**Definition**

For $1 \leq i \leq n$, we define

- $f^1[i] = f[i]$; and,
- $f^\ell[i] = f[f^{\ell-1}[i]]$ for $f[i] > 0$;
- $C(f, i) = (1 + f[i - 1], 1 + f^2[i - 1], \ldots, 1 + f^m[i - 1])$ where $f^m[i - 1] = 0$. 
There are two necessary and sufficient conditions for an integer array $f$ to be valid:

1. $f[1] = 0$ and for $2 \leq i \leq n$, we have $f[i] \in (0) \cup C(f, i)$;
2. for $i \geq 2$ and for every $j \in C(f, i)$ with $j > f[i]$, we have $f[j] \neq f[i]$.
Validation

Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>12</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f[i]$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>?</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


The candidates for $f[16]$ are in $C(f, 16) \cup (0) = (6, 4, 2, 1, 0)$.

Among these values 2 is not valid since $f[4] = 2$. 
Validation

**Theorem 2 [FGLRSSY 02]**
The validation of an array $f$ of $n$ integers can be done in $O(n)$.

**Theorem 3 [FGLRSSY 02]**
The delay (time spent on one element) is in $O(n)$. 
### Validation

#### Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w[i]$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>$f[i]$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Outline

1. Recalls
2. New results
3. Conclusions and perspectives
Recalls

New results

Conclusions and perspectives

\[ f \rightarrow \delta \]

Proposition 1

\[ \delta(0) = (1) \text{ and } \]
\[ \delta(j) = (j + 1) \cup \delta(f[j]) \cup (f[j + 1]) \text{ for } 1 \leq j < n \text{ and } \]
\[ \delta(n) = \delta(f[n]). \]
Recalls

\[ f \rightarrow \delta \]

Example

\[
\begin{array}{cccccc}
\text{\( j + 1 \)} & \cup & \delta(f[j]) & \cup & f[j + 1] & = & \delta(j) & j & f[j] \\
(1) & \cup & & \cup & (1) & = & (1) & 0 & \\
(2) & \cup & (1) & \cup & (1) & = & (2) & 1 & 0 \\
(3) & \cup & (2) & \cup & (1) & = & (3,2) & 2 & 1 \\
(4) & \cup & (1) & \cup & (1) & = & (4) & 3 & 0 \\
(5) & \cup & (2) & \cup & (1) & = & (5,2) & 4 & 1 \\
\cup & (1) & \cup & (1) & = & (1) & 5 & 0 \\
\end{array}
\]
Important

This computation is completely independent from the underlying string(s).
Assuming that $f[1..i]$ is valid, all the values for $f[i + 1]$ are in $\delta'(i) \uplus (0)$ and they do not need to be checked.

Using Proposition 1, the skeleton of the automaton is built online during the checking of the array $f$.

If $f[i + 1]$ is equal to 0, it is enough to check if the cardinality of $\delta'(i)$ is strictly smaller than the alphabet size $s$ to ensure that $f$ is valid up to position $i + 1$. 
The candidates for $f[16]$ are in $\delta'(15) \cup (0) = (6, 4, 1, 0)$. 
**Theorem 4**

The validity of a given integer array $f[1..n]$ can be checked in time and space $O(n)$. If $f$ is valid, a string for which $w$ is the border array can be computed with the same complexities.

**Theorem 5**

The delay is $O(\min\{n, \text{card } \Sigma\})$. 
Construction of all the distinct border arrays

An algorithm for generating all valid arrays becomes then obvious: all the valid candidates for $f[i]$ are in $\delta'(i - 1) \uplus (0)$. 
### Counting

<table>
<thead>
<tr>
<th>$i$</th>
<th>$B(i)$</th>
<th>$B(i, 2)$</th>
<th>$B(i, 3)$</th>
<th>$B(i, 4)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>16</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>47</td>
<td>32</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>110</td>
<td>64</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>8</td>
<td>263</td>
<td>128</td>
<td><strong>262</strong></td>
<td>263</td>
</tr>
<tr>
<td>9</td>
<td>630</td>
<td>256</td>
<td>626</td>
<td>630</td>
</tr>
<tr>
<td>10</td>
<td>1525</td>
<td>512</td>
<td>1509</td>
<td>1525</td>
</tr>
<tr>
<td>11</td>
<td>3701</td>
<td>1024</td>
<td>3649</td>
<td>3701</td>
</tr>
<tr>
<td>12</td>
<td>9039</td>
<td>2048</td>
<td>8872</td>
<td>9039</td>
</tr>
<tr>
<td>13</td>
<td>22,140</td>
<td>4096</td>
<td>21,640</td>
<td>22,140</td>
</tr>
<tr>
<td>14</td>
<td>54,460</td>
<td>8192</td>
<td>52,993</td>
<td>54,460</td>
</tr>
<tr>
<td>15</td>
<td>134,339</td>
<td>16,384</td>
<td>130,159</td>
<td>134,339</td>
</tr>
<tr>
<td>16</td>
<td>332,439</td>
<td>32,768</td>
<td>320,696</td>
<td><strong>332,438</strong></td>
</tr>
</tbody>
</table>
Number of distinct border arrays on a binary alphabet

**Proposition 2**

\[ B(n, 2) = 2^{n-1}. \]
Number of distinct border arrays on an alphabet of size $s$

**Proposition 3**

\[ B(j, s) = B(j) \text{ for } j < 2^s. \]
Proposition 4

\[ B(2^s, s) = B(2^s) - 1. \]

The missing border array has the following form:
\[ 0..2^0 - 1 \cdot 0..2^1 - 1 \cdot \ldots 0..2^{s-1} - 1. \]

It corresponds to the string \( w_s \cdot \sigma[s + 1] \) (of length \( 2^s \)) where \( w_s \) is recursively defined by:
\[ w_1 = a \quad \text{and} \]
\[ w_i = w_{i-1} \cdot \sigma[i] \cdot w_{i-1} \quad \text{for } i > 1. \]
Example

The following array $f[1..16]$ if valid on an alphabet of size at least 5:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_4[i]$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$f[i]$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>
Outline

1. Recalls
2. New results
3. Conclusions and perspectives
Conclusions

Given an integer array $f$ we can:

- say if $f$ is valid,
  - on an unbounded size alphabet or
  - on a bounded size alphabet;
- exhibit strings for which $f$ is the border array.

\[
f \leftrightarrow \delta
\]

Construct all the distinct border arrays
Get exact bounds on the number of distinct border arrays.
Perspectives

Let us recall the “failure function” of the Knuth-Morris-Pratt (1977) string matching algorithm

\[ g[j] = \max\{ i \mid \text{w[1..i − 1] suffix of w[1..j − 1] and \ w[i] \neq w[j]} \}. \]

We know that

\[ g[j] = \max\{ \delta(j - 1) - (j) \} = \max\{ \delta(f[j - 1]) - (f[j]) \}. \]