MONAA: a Tool for Timed Pattern Matching with Automata-Based Acceleration

Masaki Waga, Ichiro Hasuo, and Kohei Suenaga

National Institute of Informatics and Kyoto University
10 Apr. 2018, MT-CPS 2018

The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST, Grants-in-Aid No. 15KT0012, JSPS, and JST PRESTO Grant Number JPMJPR15E5, Japan.
Timed Pattern Matching

[Ulus et al., FORMATS’14]

**Input**
- **Time-series data** (Logs of a car/ a robot)
  e.g., The gear of a car is high at 0.1s, low at 0.3s...

- **Real-time spec.** (Spec. useful for debugging)
  e.g., The gear of a car changes too frequently

**Output**
- The intervals where the spec. is satisfied in the data
  e.g., The gear changes too frequently in 0.3s-0.7s

Timed Words
(Sequence of events with timestamps)
\[ w = \begin{array}{cccc}
0 & 0.1 & 0.3 & 0.5 & 0.7 \\
\end{array} \]

Timed Automata
[Alur & Dill, TCS’94]
\[ \mathcal{A} = \begin{array}{c}
\text{start} \\
\rightarrow s_0 \\
\leftrightarrow s_1 \\
\rightarrow s_2 \\
\rightarrow s_3
\end{array} \]

\[ \mathcal{M}(w, \mathcal{A}) = \{(t, t') \mid w\restriction_{(t,t')} \in L(\mathcal{A})\} \]
MONAA Overview

Command Line Interface (MONAA)

- Command line tool for timed pattern matching
- We can inspect a log
- The log is read lazily
  - online monitor
- Text-based I/O

C++ API (libmonnaa)

- Execute timed pattern matching in a user’s code
- Accelerated by Skipping
  [Waga et al., FORMATS’17]
- I/O by function/class
Outline

1. Algorithm in MONAA
   • Skipping for timed pattern matching
     [Waga et al., FORMATS’17]

2. Frontend of MONAA
   • Command line interface (CLI) / C++ API

3. Experiments
Skipping for String Matching

**Brute-Force Search**

Find “STRING” (pattern) from “STRONG_STRING” (target)

**KMP Search**

[Knuth+, SIAM J. Comput. '77]

Match until the 3rd char.

Table for length 3 partial match

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>*</td>
<td>S</td>
<td>R</td>
</tr>
<tr>
<td>X</td>
<td>*</td>
<td>*</td>
<td>T</td>
</tr>
<tr>
<td>✓</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Skip 2 trials!!
Brute-Force Algorithm for Timed Pattern Matching

Find $A = \text{start} \rightarrow s_0 \xrightarrow{\text{low}} s_1 \xrightarrow{\text{high, } x < 1} s_2 \xrightarrow{\$} s_3$ (pattern)

from $w =$ low high low low high low $\rightarrow t$ (target)

1st trial

2nd trial

3rd trial

$\{ t \in [0, 0.1) \}
\{ t' \in (0.3, 0.5] \}
\{ t' - t \in (0, 1) \}$

$\{ t \in [0, 0.1) \}
\{ t' \in (0.7, 1.1] \}
\{ t' - t \in (0, 1) \}$

$\{ t \in [0.3, 0.5) \}
\{ t' \in (0.7, 1.2] \}
\{ t' - t \in (0, 1) \}$
KMP-Style Algorithm for Timed Pattern Matching

Find \( A = \) start \( \rightarrow s_0 \xrightarrow{low} s_1 \xrightarrow{\text{high}} s_2 \xrightarrow{\$, x < 1} s_3 \) (pattern)

from \( w = \)

\[
\begin{array}{cccccc}
\text{low} & \text{high} & \text{low} & \text{high} & \text{low} \\
0 & 0.1 & 0.3 & 0.5 & 0.7 & 1.2 \\
\end{array}
\]

\( t \)

1st trial

Skip 1 trial!!

2nd trial

2nd trial

\[
\begin{align*}
\{ t \in [0, 0.1) \\
t' \in (0.3, 0.5] \\
t' - t \in (0, 1) \}
\end{align*}
\]

\[
\begin{align*}
\{ t \in [0, 0.1) \\
t' \in (0.7, 1.1] \\
t' - t \in (0, 1) \}
\end{align*}
\]

\[
\begin{align*}
\{ t \in [0.3, 0.5) \\
t' \in (0.7, 1.2] \\
t' - t \in (0, 1) \}
\end{align*}
\]
Problems in Skipping for Timed Pattern Matching

- The length of partial match is unbounded.

### String Matching

<table>
<thead>
<tr>
<th>Length 3 partial match in string matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
</tr>
</tbody>
</table>

### Timed Pattern Matching

- Infinitely many timestamps

<table>
<thead>
<tr>
<th>Infinitely Many Tables!!</th>
</tr>
</thead>
<tbody>
<tr>
<td>We construct for each length</td>
</tr>
</tbody>
</table>

- Infinitely many timestamps

<table>
<thead>
<tr>
<th>Length 2 partial match in timed pattern matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Infinitely many timestamps
Problems in Skipping for Timed Pattern Matching

- The length of partial match is unbounded.

### String Matching

<table>
<thead>
<tr>
<th>Length 3 partial match in string matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>S T R</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>✖</td>
</tr>
<tr>
<td>✔</td>
</tr>
</tbody>
</table>

### Timed Pattern Matching

- Infinitely many timestamps

#### Table for state $s_2$ partial match

<table>
<thead>
<tr>
<th>State $s_2$ partial match in timed pattern matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>start $s_0$ low $s_1$ high $s_2$ low, $x &lt; 1$</td>
</tr>
</tbody>
</table>

Represent by a timed automaton
Problems in Skipping for Timed Pattern Matching

- The length of partial match is unbounded.

Table for **length 3** partial match

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>T</th>
<th>R</th>
<th>I</th>
<th>N</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>*</td>
<td>S</td>
<td>T</td>
<td>I</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>*</td>
<td>*</td>
<td>S</td>
<td>T</td>
<td>R</td>
<td>I</td>
</tr>
<tr>
<td>✓</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>S</td>
<td>T</td>
<td>R</td>
</tr>
</tbody>
</table>

- Infinitely many timestamps

**Length 3** partial match in string matching

```
S T R
```

Represent by a timed automaton

**State s2** partial match in timed pattern matching

```
start → s0 (low) → s1 → s2
high, x < 1
low, x < 1
```
Skipping for Timed Pattern Matching

Table for state $s_2$ partial match

We want to compare them
Skipping for Timed Pattern Matching

Table for **state** $s_2$ partial match

![Diagram showing transitions and states for skip patterns.](image-url)
Emptiness Checking by Zone Construction

Labelled by a set of “similar” clock valuations represented by a zone

Thm. (soundness and completeness)
Zone automata maintain state reachability.
Outline

1. Algorithm in MONAA
   - Skipping for timed pattern matching
     [Waga et al., FORMATS’17]

2. Frontend of MONAA
   - Command line interface (CLI) / C++ API

3. Experiments
Input of MONAA (CLI)

digraph G {
  1 [init=1][match=0];
  2 [init=0][match=0];
  3 [init=0][match=0];
  4 [init=0][match=0];
  5 [init=0][match=0];
  6 [init=0][match=0];
  7 [init=0][match=0];
  1->2 [label=b][reset=\"{}\"];
  2->3 [label=a][guard=\"\{x0 < 1\}\"];
  3->4 [label=a][guard=\"\{x0 < 1\}\"];
  4->5 [label=a][guard=\"\{x0 < 1\}\"];
  5->6 [label=a][guard=\"\{x0 < 1\}\"];
  6->6 [label=a][guard=\"\{x0 < 1\}\"];
  6->7 [label=a][guard=\"\{x0 > 1\}\"];
  }

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.267718</td>
<td>0.280545</td>
</tr>
<tr>
<td>b</td>
<td>0.293307</td>
<td>0.300000</td>
</tr>
<tr>
<td>b</td>
<td>0.306016</td>
<td>0.306016</td>
</tr>
<tr>
<td>b</td>
<td>0.318685</td>
<td>0.331324</td>
</tr>
<tr>
<td>b</td>
<td>0.343941</td>
<td>0.356541</td>
</tr>
<tr>
<td>b</td>
<td>0.356541</td>
<td>0.369126</td>
</tr>
<tr>
<td>b</td>
<td>0.381700</td>
<td>0.394264</td>
</tr>
<tr>
<td>b</td>
<td>0.400000</td>
<td>0.406823</td>
</tr>
<tr>
<td>b</td>
<td>0.411769</td>
<td>0.419377</td>
</tr>
<tr>
<td>b</td>
<td>0.421929</td>
<td>0.444483</td>
</tr>
<tr>
<td>b</td>
<td>0.444485</td>
<td>0.457039</td>
</tr>
<tr>
<td>b</td>
<td>0.457062</td>
<td>0.469615</td>
</tr>
<tr>
<td>b</td>
<td>0.469677</td>
<td>0.482231</td>
</tr>
<tr>
<td>b</td>
<td>0.482231</td>
<td>0.482231</td>
</tr>
</tbody>
</table>
Output of MONAA (CLI)

Masakis-MacBook-Pro:examples calros$ monaa -f torque.dot < torque-1000.txt

<table>
<thead>
<tr>
<th>Lower Bound</th>
<th>Condition</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>137.700380</td>
<td>( t &lt; 137.734850 )</td>
<td></td>
</tr>
<tr>
<td>138.734850</td>
<td>( t' \leq 138.744730 )</td>
<td></td>
</tr>
<tr>
<td>1.000000</td>
<td>( t' - t \leq 1.044350 )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lower Bound</th>
<th>Condition</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>695.670550</td>
<td>( t &lt; 695.683090 )</td>
<td></td>
</tr>
<tr>
<td>696.683090</td>
<td>( t' \leq 696.684680 )</td>
<td></td>
</tr>
<tr>
<td>1.000000</td>
<td>( t' - t \leq 1.014130 )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lower Bound</th>
<th>Condition</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>842.300000</td>
<td>( t &lt; 842.309420 )</td>
<td></td>
</tr>
<tr>
<td>843.309420</td>
<td>( t' \leq 843.315490 )</td>
<td></td>
</tr>
<tr>
<td>1.000000</td>
<td>( t' - t \leq 1.015490 )</td>
<td></td>
</tr>
</tbody>
</table>
C++ API (libmonaa)

User’s Program

- Get feedback
- Give Logs

C++ API (libmonaa)

- e.g., online runtime verification
- Execute timed pattern matching in a user’s code
- Accelerated by Skipping
- I/O by function/class

M. Waga (NII)
Better Performance by libmonaa

MONAA

libmonaa \(\rightarrow\) monaa \(\rightarrow\) Exec.

TA/TRE

Timed Word

libmonaa (TA is hard-coded)

libmonaa \(\rightarrow\) Monitor \(\rightarrow\) Exec.

TA

Timed Word

Compile

More optimization!!
Outline

1. Algorithm in MONAA
   • Skipping for timed pattern matching
     [Waga et al., FORMATS’17]

2. Frontend of MONAA
   • Command line interface (CLI) / C++ API

3. Experiments
Comparison with Montre

**MONAA**
- Use timed automata
- We can also construct a TA from a TRE
- Accelerated by **skipping**
- Both command line and C++ interface

**Existing Tool (Montre)**
- Use timed regular expression
- (online) On-the-fly construction of a state machine from TRE
- Only command line interface

Our algorithm should work faster because of skipping

[Ulus, CAV'17]
Experiments

- Monitoring of a Simulink model of an automatic transmission in a car.

- Model and spec. are from an automotive benchmark paper [Hoxha et al., ARCH ’15]

- Events: gears: $g_1,g_2,g_3,g_4$
  - velocity: $v > \bar{v}, v \leq \bar{v}$
  - RPM: $\omega > \bar{\omega}, \omega \leq \bar{\omega}$
Experiments

Gear changes from $g_1$ to $g_4$ in 10 sec. and RPM changes to high enough, but velocity is still low.
### Table 1. Execution time (sec.)

<table>
<thead>
<tr>
<th>Length of timed word</th>
<th>MONAA (TRE)</th>
<th>MONAA (TA)</th>
<th>libmonaa (TA is hard coded)</th>
<th>Montre (online)</th>
<th>Montre (offline)</th>
</tr>
</thead>
<tbody>
<tr>
<td>306</td>
<td>7.03</td>
<td>0.80</td>
<td>0.20</td>
<td>0.13</td>
<td>0.03</td>
</tr>
<tr>
<td>127,552</td>
<td>7.55</td>
<td>1.27</td>
<td>0.31</td>
<td>37.45</td>
<td>1.56</td>
</tr>
<tr>
<td>255,750</td>
<td>8.05</td>
<td>1.73</td>
<td>0.42</td>
<td>75.93</td>
<td>3.13</td>
</tr>
<tr>
<td>383,168</td>
<td>8.54</td>
<td>2.21</td>
<td>0.53</td>
<td>115.88</td>
<td>4.69</td>
</tr>
<tr>
<td>508,756</td>
<td>9.16</td>
<td>2.69</td>
<td>0.64</td>
<td>153.71</td>
<td>6.21</td>
</tr>
<tr>
<td>632,484</td>
<td>9.53</td>
<td>3.14</td>
<td>0.75</td>
<td>189.55</td>
<td>7.75</td>
</tr>
<tr>
<td>758,500</td>
<td>10.05</td>
<td>3.60</td>
<td>0.85</td>
<td>216.92</td>
<td>9.33</td>
</tr>
<tr>
<td>894,692</td>
<td>10.53</td>
<td>4.06</td>
<td>0.97</td>
<td>260.77</td>
<td>10.88</td>
</tr>
<tr>
<td>1,011,426</td>
<td>11.05</td>
<td>4.56</td>
<td>1.07</td>
<td>289.63</td>
<td>12.39</td>
</tr>
</tbody>
</table>

- **Efficient and online**
- **Blow up!!**
- **Efficient but offline only**
Conclusion

- MONAA can inspect logs with timestamps
  - fast (skipping)
  - simple (text-based I/O)
  - flexible (C++ API)
Future Works

• Theoretical Side:

  • Investigate other techniques for efficient monitoring

• Practical Side:

  • Case study of timed pattern matching

https://github.com/MasWag/monaa