QMine: A Framework for Mining Quantitative Regular Expressions from System Traces

Pradeep Kumar Mahato, Apurva Narayan
Department of Computer Science
The University of British Columbia
BC, Canada
Motivation

Data is everywhere!!

Modern-day software systems are complex and produce an exorbitant amount of data. Understanding system behaviour using these data is crucial.

Common datasources:
- Medical instruments
- Geolocation devices
- Heavy machinery and their controllers

Fig: Sample heartbeat record

Motivation

Temporal properties provide information about the occurrence of events. It draws insights from the system specification.

Challenges
- Extracting such properties is challenging
- Specification are sometimes loosely specified
- Lack of proper specifications

Fig: Stock market dashboard sample
Objective

Mine for temporal properties in the form of quantitative regular expressions from system traces.

Few major contributions of our work

- Present novel framework to mine temporal properties using Quantitative Regular Expressions
- Justify algorithmic characterization with space and time
- Validate using industrial system traces
Outline

- Background
- Methodology
- Experiments
- Conclusion
- Future Work
Background
Event, traces and their representation

Event
The alphabet of events is a finite alphabet of strings

Trace
Group of events forms a trace

Notations

Events: $\alpha_i$
Global alphabet set: $\Sigma$
Trace contains $\langle \alpha_1, \alpha_2, \cdots, \alpha_n \rangle \in \Sigma$

Fig: Sample log from Xorg in ubuntu
Quantitative Regular Expressions

Proposed by Rajeev Alur et al., Quantitative Regular Expressions (QRE) are representations to evaluate quantitative values using regular expressions defined over an input domain $D$ and cost domain $C$.

**QRE expression for performing average:**

$$a = \text{atom}(x \rightarrow x.\text{type} = M, x \rightarrow x.\text{val})$$

$$b = \text{atom}(x \rightarrow x.\text{type} = M, x \rightarrow 1)$$

$$\text{num} = \text{iter}(a, a, (x, y) \rightarrow x + y)$$

$$\text{den} = \text{iter}(b, b, (x, y) \rightarrow x + y)$$

$$\text{avg} : QRE\langle X, Y \rangle = \text{combine}(\text{num}, \text{den}, (x, y) \rightarrow x/y)$$

$X$ and $Y$ are input and output data types respectively.
Quantitative Regular Expression Template (QRET)

**Definition 4** (QRET). A QRET is a valid template if it contains at least two events \((\alpha_i, \alpha_j) \in \Sigma\) and a series of \(m\) quantitative values \(q_1 \cdots q_m\) bounded by \(\alpha_i\) and \(\alpha_j\).

More precisely, the most rudimentary form of QRET can be explained as \(QRET = \alpha_1 \mathbb{R} \alpha_2\), where \(\mathbb{R}\) is a set of real numbers.

**Sample**

\(QRET : 0 \ M \ 1\)

\(Event_0\) followed by a quantitative value, \(M\), which is then followed by \(Event_1\).

Placeholder 0 and 1 are replaced by all events in the alphabets \(\langle \alpha_0, \alpha_1, \cdots \rangle \in \Sigma\).
QTrace: A trace complaint of QRET instance

A series of events with quantitative values between them forms a QTrace file.

Formally, each quantitative value (\( \mathbb{R} \)) is enclosed with starting event \( \alpha_i \) and closing event \( \alpha_{i+1} \)

Sample

\( \alpha_a \) 123 \( \alpha_b \): Quantitative value 123 bounded by \( \alpha_a \) and \( \alpha_b \)

Fig: Sample illustration of Arrhythmia Dataset
Methodology
QMine Framework

**QMine Core** uses parallelization for mining

Results are stored in **Mine Map** for future query.

This avoids re-mining

*Fig: QMine framework*
QMine Algorithm

**Algorithm 1:** QMine(QRET, QTrace, $\Sigma$, $\xi$)

- **Input:** QRET, QTrace, alphabet $\Sigma$, mine-map $\xi$
- **Result:** Mined properties for QTrace

1. Generate all permutations of QRET from $\Sigma$
2. Initialize Finite State Acceptor (FSA)
3. Set all threads with $\xi$ and the FSA
4. Divide QTrace segments per thread
5. **foreach** segment $\in$ QTrace segments **do**
   6. **mineInternal** (segment, $\xi$, FSA)
7. **end**

**Algorithm 2:** mineInternal(segment, $\xi$, FSA)

- **Input:** QTrace Segment, $\xi$, FSA
- **Result:** Mined property satisfied by a QTrace segment

1. action EVENT
2. record patterns for QTrace Segment match
3. action NUM
4. push digit to $\xi$
5. action FINISH
6. match overall pattern
7. push $\xi$ into global shared space & increment count
8. return TRUE // Accepting state
9. action ERROR
10. remove inserted values from $\xi$
11. return FALSE // Error State

**Fig:** QMine algorithm
QMine Threading

Fig: Thread workload distribution
Experiments
Heartbeat analysis for arrhythmia

**Dataset**  Arrhythmia patients  
**Source**  UCI Machine Learning Repository[^2]

**Patients**  452  
**Classes**  15

- Class 1  Normal reading  
- Class 2  Patient with coronary artery diseases  
- Class 3 - 15  Different groups for arrhythmia patients

**Events (in consideration)**  Age, Sex, Height, Weight, Heart Rate, and Class

**Objective**  Identify patterns for arrhythmia patients

**QRET**  0 M 1 M 2 M 3 M 4 M 5 M 6  
**Alphabet Set Size**  7  
**Mining Combination**  7 !  i.e  5040 total patterns
### Heartbeat analysis for arrhythmia

<table>
<thead>
<tr>
<th>Age</th>
<th>Sex</th>
<th>Count</th>
<th>Height (cm)</th>
<th>Weight (kg)</th>
<th>Heart Rate (bpm)</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-Male</td>
<td></td>
<td>Mean</td>
<td>Variance</td>
<td>Mean</td>
<td>Variance</td>
</tr>
<tr>
<td>0-39</td>
<td></td>
<td>0</td>
<td>36</td>
<td>174.194</td>
<td>24.8789</td>
<td>77.8333</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>118</td>
<td>159.356</td>
<td>14.3987</td>
<td>60.8305</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>5</td>
<td>167.2</td>
<td>5.36</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>12</td>
<td>161.333</td>
<td>65.2222</td>
<td>62.1667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>13</td>
<td>168.154</td>
<td>6.74556</td>
<td>69.3846</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>28</td>
<td>162.5</td>
<td>42.25</td>
<td>61.6429</td>
</tr>
<tr>
<td>40-99</td>
<td></td>
<td>0</td>
<td>123</td>
<td>171.407</td>
<td>42.5665</td>
<td>76.0569</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>183</td>
<td>160.541</td>
<td>25.9532</td>
<td>68.0383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>29</td>
<td>169.103</td>
<td>40.9893</td>
<td>80.8276</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>40</td>
<td>158.7</td>
<td>20.61</td>
<td>73.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>115</td>
<td>170.13</td>
<td>43.7656</td>
<td>76.1043</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>77</td>
<td>157.987</td>
<td>21.2596</td>
<td>71.2208</td>
</tr>
</tbody>
</table>

Table: Summary of quantitative values from arrhythmia dataset
Synthetic analysis (stress testing)

**Dataset**  Self generated adhering to QRET pattern

**CPU**  : Intel i7-2630 QM (2.6 Ghz max boost)

**Max Threads**  16

**FSM framework**  Ragel [3]

**Events (in consideration)**  < variable >

**QRET**  < variable >

**Alphabet Set Size**  < variable >

**Mining Combination**  < variable >
Synthetic analysis (stress testing)

<table>
<thead>
<tr>
<th>QRET</th>
<th>Alphabet Size</th>
<th>Total QRE Instances</th>
<th>Compile Time Taken (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0M1</td>
<td>10</td>
<td>90</td>
<td>2,840</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>2450</td>
<td>2,955</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>249500</td>
<td>61,274</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>999000</td>
<td>247,728</td>
</tr>
<tr>
<td>0M1M2</td>
<td>10</td>
<td>720</td>
<td>2,912</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>6840</td>
<td>3,258</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>117600</td>
<td>23,114</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>970200</td>
<td>638,822</td>
</tr>
</tbody>
</table>

Table: Analysis with QRET pattern, alphabet size and compilation time
Synthetic analysis (stress testing)

Table: Analysis with varying alphabet size and event placeholders
Algorithmic complexity

<table>
<thead>
<tr>
<th>Complexity Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
</tr>
<tr>
<td><strong>Space</strong></td>
</tr>
<tr>
<td><strong>Execution</strong></td>
</tr>
</tbody>
</table>

* placeholder length denoted by $\tau$
Conclusion

- We presented a novel QMine framework for extracting and inspecting for interesting patterns
- Scalability of the framework is almost linear*
- Our algorithm is robust, sound and complete

Few of the future works could be:
- Improve mining with hard constraints
- Dynamic pattern inspection based on system behaviour

* Once the FSM model is generated, our mining is almost linear via parallelization
References

Thank you