Mining Patterns and Building Classifiers From Software Data: Addressing Soft. Maintenance & Reliability Issues

David Lo
School of Information Systems
Singapore Management University

Presentation at UIUC
July 31, 2009
Motivation: Maintenance Issues

- **Maintenance**: Update to an existing software
  - Need to understand how a software behaves

- **Specification**: Description on what a software is supposed to behave
  - Locking Protocol: `<mutex_lock, mutex_unlock>`
  - JTA Protocol [JTA]: `<TxManager.begin, TxManager.commit>`, etc.
  - Telecommunication Protocol [ITU]:
    - `<off_hook, dial_tone_on, dial_tone_off, seizure_int, ring_tone, answer, connection_on>`
  - JAAS Authentication Enforcer Strategy Pattern [SNL06]:
    - `<Subject.getPrincipal, PrivilegedAction.create, Subject.doAsPrivileged, JAAS_Module.invoke, Policy.getPermission, Subject.getPublicCredential, PrivilegedAction.Run>`
Motivation: Maintenance Issues

- Existing problems in specification: Lack, incomplete and outdated specifications [LK06, ABL02, YEBBD06, DSB04, etc.]

- Cause difficulty in understanding an existing system

- Contributes to high software cost
  - Prog. maintenance: 90% of soft. cost [E00, CC02]
  - Prog. understanding: 50% of maint. cost [S84, CC02]
  - US GDP software component: $214.4 billion [US BEA]

- Solution: Specification Discovery
Motivation: Reliability Issues

- We depend on correct working of software systems
  - Banking application, control systems, etc
- Software bugs have caused a lot of issues
  - 59.5 billion dollars lost to US economy annually [NIST'2002]
  - Privacy & security issues
- Much savings could be made by either
  - Preventing bugs
  - Detecting failures
  - Localizing bugs
  - Suggesting fix
  - Guaranteeing no bugs could ever exist
  - Healing failures (e.g., Microsoft Shims), etc.
Can Data Mining Help?

YES!
Outline

- **Software Specification Discovery**
  - Semantics based on standard software specifications
  - Closed pattern mining strategy
  - Performance study and case study
  - Addressing “lack of specifications” problem

- **Classification of software behaviors**
  - Sequential pattern-based classification
  - Improving efficiency & accuracy
  - Application to detect failures from software data
  - Addressing reliability of systems
Efficient Mining of Iterative Patterns for Software Specification Discovery

David Lo†
Joint work with: Siau-Cheng Khoo† and Chao Liu‡

†Prog. Lang. & Sys. Lab
Department of Computer Science
National Uni. of Singapore

‡Data Mining Group
Department of Computer Science
Uni. of Illinois at Urbana-Champaign
Our Specification Discovery Approach

- Analyze **program execution traces**
- Discover **patterns of program behavior**, e.g.:
  - Locking Protocol [YEBBD06]: \(<lock, unlock>\)
  - Telecom. Protocol [ITU], etc.
- **Address unique nature of prog. traces:**
  - Pattern is repeated across a trace
  - A program generates different traces
  - Interesting events might not occur close together
Need for a Novel Mining Strategy

- **Sequential Pattern Mining** [AS95, YHA03, WH04] - A series of events (itemsets) supported by (i.e. sub-sequence of) a significant number of sequences.
  
  **Required Extension:** Consider multiple occurrences of patterns in a sequence

- **Episode Mining** [MTV97, GO3] - A series of closely-occurring events recurring frequently within a sequence.
  
  **Required Extension:** Consider multiple sequences; Remove the restriction of events occurring close together.
Iterative Patterns – Semantics

- A series of events supported by a significant number of instances:
  - Repeated within a sequence
  - Across multiple sequences.

- Follow the semantics of Message Seq. Chart (MSC) [ITU] and Live Seq. Chart (LSC) [DH01].

- Describe constraints between a chart and a trace segment obeying it:
  - Ordering constraint [ITU,KHPLB05]
  - One-to-one correspondence [KHPLB05]
Iterative Patterns – Semantics

Switching Sys

<table>
<thead>
<tr>
<th>Calling Party</th>
<th>Called Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>off_hook</td>
<td></td>
</tr>
<tr>
<td>dial_tone_on</td>
<td></td>
</tr>
<tr>
<td>dial_tone_off</td>
<td></td>
</tr>
<tr>
<td>seizure</td>
<td></td>
</tr>
<tr>
<td>ack</td>
<td></td>
</tr>
<tr>
<td>ring_tone</td>
<td></td>
</tr>
<tr>
<td>answer</td>
<td></td>
</tr>
<tr>
<td>connection_on</td>
<td></td>
</tr>
</tbody>
</table>

o TS1: off_hook, seizure, ack, ring_tone, answer, ring_tone, connection_on

o TS2: off_hook, seizure, ack, ring_tone, answer, answer, answer, connection_on

o TS3: off_hook, seizure, ack, ev1, ring_tone, ev1, answer, connection_on

[ITU]
Iterative Patterns – Semantics

- Given a pattern $P (e_1 e_2 \ldots e_n)$, a substring $SB$ is an instance of $P$ iff
  
  $$SB = e_1;[-e_1, \ldots, e_n]*;e_2;\ldots;[-e_1, \ldots, e_n]*;e_n$$

- Pattern: `<off_hook, seizure, ring_tone, answer, connection_on>`
- $S_1$: off_hook, ring_tone, seizure, answer, connection_on  
- $S_2$: off_hook, seizure, ring_tone, answer, answer, answer, connection_on  
- $S_3$: off_hook, seizure, ev1, ring_tone, ev1, answer, connection_on  
- $S_4$: off_hook, seizure, ev1, ring_tone, ev1, answer, connection_on, off_hook, seizure_int, ev2, ring_tone, ev3, answer, connection_on
Mining
Algorithm
Projected Database Operations

- Projected-all of SeqDB wrt pattern P -
  \[ SeqDB_{all}^P \]
  
  **Return:** All suffixes of sequences in SeqDB where for each, its infix is an instance of pattern P

- Support of a pattern = size of its proj. DB

- SeqDB_{all}^{ev} is formed by considering occurrences of ev

- SeqDB_{all}^{p++,ev} can be formed from SeqDB_{all}^{p}
Pruning Strategies

Apriori Property
If a pattern \( P \) is not frequent, \( P^{++} \) evs can not be frequent.

Closed Pattern
Definition: A frequent pattern \( P \) is closed if there exists no super-sequence pattern \( Q \) where:
- \( P \) and \( Q \) have the same support
- and corresponding instances

Sketch of Mining Strategy
1. Depth first search
2. Cut search space of non-frequent and non-closed patterns
**Closure Checks and Pruning – Definitions**

- **Prefix, Suffix Extension (PE) (SE)**
  - An event that can be added as a prefix or suffix (of length 1) to a pattern resulting in another with the same support.

- **Infix Extension (IE)**
  - An event that can be inserted as an infix (one or more times) to a pattern resulting in another with the same support and corresponding instances.

<table>
<thead>
<tr>
<th></th>
<th>Pattern: \langle A, C \rangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>\langle X, A, B, B, C, D \rangle</td>
</tr>
<tr>
<td>S2</td>
<td>\langle X, A, B, B, C, D, E, F, G \rangle</td>
</tr>
<tr>
<td>S3</td>
<td>\langle B, C, A, D, E, D \rangle</td>
</tr>
</tbody>
</table>

- Prefix Ext: \{\langle X \rangle\}
- Suffix Ext: \{\langle D \rangle\}
- Infix Ext: \{\langle B \rangle\}
Closure Checks and Pruning - Theorems

- **Closure Checks**: If a pattern \( P \) has no (PE, IE and SE) then it is closed otherwise it is not closed.

- **InfixScan Pruning Property**: If a pattern \( P \) has an IE and \( IE \notin SeqDB_p \), then we can stop growing \( P \).

---

<table>
<thead>
<tr>
<th></th>
<th>Pattern</th>
<th>Prefix Ext</th>
<th>Infix Ext</th>
<th>Suffix Ext</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>(&lt;X,A,B,B,C,D&gt;)</td>
<td>{(&lt;X&gt;)}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>(&lt;X,A,B,B,C,D,E,F,G&gt;)</td>
<td></td>
<td>{(&lt;B&gt;)}</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>(&lt;B,C,A,D,E,D&gt;)</td>
<td></td>
<td>{(&lt;D&gt;)}</td>
<td></td>
</tr>
</tbody>
</table>

\(<A,C>\) is not closed and we can stop growing it. No need to check for \(<A,C,\ldots>\)
Procedure MinePatterns
Inputs:
$SeqDB$ : Sequence DB, $min\_sup$ : Min. Sup. Thresh.
Methods:
1: Let $Freq =$ Frequent length-1 patterns
2: For every $f\_ev$ in $Freq$
3: Call MineRecurse $(f\_ev, SeqDB^{all}_{f\_ev}, min\_sup, Closed, Freq)$

Procedure MineRecurse
Inputs:
$Pat$ : Pattern so far, $SeqDB^{all}_{Pat}$ : Sequence DB
$min\_sup$ : Min. Sup. Thresh., $EV$ : Frequent Events
Methods:
4: If $(Pat$ has no extensions)
5: Output $Pat$
6: For every $f\_ev$ in $\{e|e \in EV \land (sup(Pat++e) \geq min\_sup)\}$
7: Let $NxtPat = Pat++f\_ev$
8: If $(\forall e. (e \in InfixExt(NxtPat) \land e \notin SeqDB^{all}_{NxtPat}))$
9: Call MineRecurse $(NxtPat, ProjDB, min\_sup, Closed, EV)$
Performance & Case Studies
Performance Study - I

- **Synthetic Dataset**
  - IBM Simulator: D5C20N10S20
Performance Study - II

- **Dataset Gazelle (KDD Cup – 2000)**
  - Click stream datasets

![Runtime (s) - (log-scale)](image1)

![Patterns - (log-scale)](image2)
Performance Study - III

- **Dataset TCAS**
  - Program traces from Siemens dataset - commonly used for benchmark in error localization
Case Study

- **JBoss App Server** – Most widely used J2EE server
  - A large, industrial program: more than 100 KLOC
  - Analyze and mine behavior of transaction component of JBoss App Server

- **Trace generation**
  - Weave an instrumentation aspect using AOP
  - Run a set of test cases
  - Obtain 28 traces of 2551 events and an average of 91 events

- **Mine using min_sup set at 65% of the |SeqDB|** – 29s vs >8hrs
Case Study

- Post-processings & Ranking – 44 patterns
- Top-ranked patterns correspond to interesting patterns of software behavior:
  - <Connection Set Up Evs, Tx Manager Set Up Evs, Transaction Set Up Evs, Transaction Commit Evs (Transaction Rollback Evs), Transaction Disposal Evs>

  Top Longest Patterns
  - <Resource Enlistment Evs, Transaction Execution Evs, Transaction Commit Evs (Transaction Rollback Evs), Transaction Disposal Evs>
  - <Lock-Unlock Evs>

Most Observed Pattern
| Connection Set Up | Transaction Manager Locator getInstance  
|                 | TransactionManagerLocator.locate  
|                 | TransactionManagerLocator.tryJNDI  
|                 | TransactionManagerLocator.usePrivateAPI  
| TxManager Set Up | TxManager.begin  
|                 | XidFactory newXid  
|                 | XidFactory getNextId  
|                 | XidImpl.getTrulyGlobalId  
| Transaction Set Up | TransactionImpl.associateCurrentThread  
|                 | TransactionImpl.getLocalId  
|                 | XidImpl.getLocalId  
|                 | LocalId.hashCode  
|                 | TransactionImpl.equals  
|                 | TransactionImpl.getLocalIdValue  
|                 | XidImpl.getLocalIdValue  
|                 | TransactionImpl.getLocalIdValue  
|                 | XidImpl.getLocalIdValue  
| Transaction Commit | TxManager.commit  
|                 | TransactionImpl.commit  
|                 | TransactionImpl.beforePrepare  
|                 | TransactionImpl.checkIntegrity  
|                 | TransactionImpl.checkBeforeStatus  
|                 | TransactionImpl.endResources  
|                 | TransactionImpl.completeTransaction  
|                 | TransactionImpl.cancelTimeout  
|                 | TransactionImpl.doAfterCompletion  
|                 | TransactionImpl.instanceDone  
| Transaction Disposal | TxManager.releaseTransactionImpl  
|                 | TransactionImpl.getLocalId  
|                 | XidImpl.getLocalId  
|                 | LocalId.hashCode  
|                 | LocalId.equals  

Longest Iter. Pattern from JBoss Transaction Component
Library Usage Rules & Bug Detection: Windows Application -- Extension

- Collect traces from **10 Windows Application**:
  - Excell, OneNote, TextPad, VS.Net, Visio, WMPlayer, Virtual PC, Movie Maker, WordPad, Access

- Collect traces pertaining to:
  - Registry, Memory Management, GDI (Device Control and UI related API)
  - Produces **several million events**
Library Usage Rules & Bug Detection: Windows Application -- Extension

V HeapAlloc(,,); -> HeapFree(,,V);
V GlobalAlloc(,); -> GlobalFree(V);
V VirtualAlloc(,,); -> VirtualFree (,,V);

....

HeapFree(,,V); -P> V HeapAlloc(,,);

Detect double free, which is disallowed
“Calling HeapFree twice with the same pointer can cause heap corruption, resulting in subsequent calls to HeapAlloc returning the same pointer twice.” [MSDN]
Library Usage & Bug Detection:  
Windows Application -- Extension

RegCreateKeyExA(V, .) -> RegCloseKey(V);  
Not all opened registry need to be closed  
Predefined keys need not be closed

V CreateCompatDC(); -> DeleteDC(V);  
V CreateCompatBmap(,,); -> DeleteObj(V);  
V CreateRectRgn(,,,)-> DeleteObj(V);  
DeleteDC(V) -precede-> V CreateCompDC()  
SetBkColor(,V); -> V SetBkColor(,)  
...

### Lsc Start chat

<table>
<thead>
<tr>
<th>0:RTree</th>
<th>0:Jeti</th>
<th>0:ChatWin</th>
<th>0:JID</th>
<th>1:JID</th>
<th>0:Backend</th>
<th>0:Connect</th>
<th>0:Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- chat(...)
- chat(...)
- createThread(…)
- getUser()
- getResource(…)
- send(…)
- send(…)

### Lsc Draw shape

<table>
<thead>
<tr>
<th>0:Mode</th>
<th>0:PictChat</th>
<th>0:JID</th>
<th>0:PictHistory</th>
<th>0:Backend</th>
<th>0:Connect</th>
<th>0:Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- getMyJID()
- draw(...)
- toString(…)
- addShapeDrawnByMe(…)
- send(…)
- send(…)
- send(…)
- send(…)

**Mined LSCs - Jeti Mess. App**
Visualization in IBM RSA ▲▼ Violation Trace - Scenario Based Test

E: 1180527437140 75: jabber.Backend.send(Packet)
B: jeti.msdaspects.MUSDAspectJetiTest01[57] lifeline 1 <- jabber.Backend@2bee2bee
B: jeti.msdaspects.MUSDAspectJetiTest01[57] lifeline 0 <- shapes.PictureChat@2bdc2bdc
C: jeti.msdaspects.MUSDAspectJetiTest01[57] (1,1,0,0) Cold
E: 1180527437140 76: jabber.Backend.send(Packet)
B: jeti.msdaspects.MUSDAspectJetiTest01[57] lifeline 2 <- shapes.PictureHistory@76687668
C: jeti.msdaspects.MUSDAspectJetiTest01[57] (1,2,1,0) Hot
F: jeti.msdaspects.MUSDAspectJetiTest01[57] Violation
Classification of Software Behaviors for Failure Detection: A Discriminative Pattern Mining Approach

David Lo¹, Hong Cheng², Jiawei Han³, Siau-Cheng Khoo⁴, and Chengnian Sun⁴

¹Singapore Management University, ²Chinese University of Hong Kong, ³University of Illinois at Urbana-Champaign, ⁴National University of Singapore
Software, Its Behaviors and Bugs

- Software is ubiquitous in our daily life
- Many activities depend on correct working of software systems
- Program behaviors could be collected
  - An execution trace: a sequence of events
  - A path that programs take when executed
  - A program contains many behaviors
  - Some correspond to good ones, others to bad ones
- Bugs have caused the loss of billions of dollars (NIST report)
Can Data Mining Help?

- Pattern mining tool for program behaviors
- Recent development of the pattern-based classification approach

In this work, we extend the above work to:
- Propose a new pattern definition which could be more efficiently mined (closed unique iterative pattern)
- Develop a new pattern-based classification on sequential data (iter. pattern-based classification)
- Apply the above to detection of bad behaviors in software traces for failure detection
Our Goal

“Based on historical data of software and known failures, we construct a pattern-based classifier working on sequential software data to generalize the failures and to detect unknown failures.”

- Failure detection is the first step/building block in software quality assurance process.
- Could be chained/integrated with other work on:
  - Bug localization
  - Test case augmentation
  - Bug/malware signature generation
Usage Scenarios

\(<e_1,e_2,e_3,e_4...e_n>\)

Unknown Trace or Sequence of Events

Trained Sequential Classifier

Discriminative Features

Normal

Failure

Test Suite Augmentation Tool → Failure Detector

Failure Detector → Fault Localization
Related Studies

- Lo et al. has proposed an approach to mine for iterative patterns capturing series of events appearing within a trace and across many traces. (LKL-KDD’07)

- Cheng et al., Yan et al. have proposed a pattern based classification method on transaction and graph datasets. (CYHH-ICDE’07, YCHY-SIGMOD’08)
Research Questions

- How to build a pattern-based classifier on sequential data which contains many repetitions?
- How to ensure that the classification accuracy is good?
- How to improve the efficiency of the classifier building process?
Software Behaviors & Traces

- Each trace can be viewed as a sequence of events
- Denoted as \( <e_1, e_2, e_3, ..., e_n> \)
- An event, is a unit behavior of interest
  - Method call
  - Statement execution
  - Basic block execution in a Control Flow Graph (CFG)
- Input traces \( \rightarrow \) a sequence database
Overall View of The Pattern-Based Classification Framework

- Sequence Database
- Iterative Pattern Mining
- Feature Selection
- Classifier Building
- Classifier
- Failure Detection
Iterative Patterns

- A pattern is a series of events \( (P=\langle p_1, p_2, \ldots, p_n \rangle) \)
- Given a pattern \( P \) and a sequence database \( DB \), instances of \( P \) in \( DB \) could be computed
- Based on MSC & LSC (software spec. formalisms)
- Given a pattern \( P = (e_1e_2\ldots e_n) \), a substring \( SB \) is an instance of \( P \) iff
  \[ SB = e_1;[-e_1,\ldots,e_n]^*;e_2;\ldots;[-e_1,\ldots,e_n]^*;e_n \]
- Goal: Find patterns whose instances appear often within a sequence and across multiple sequences (above a \( \text{min\_sup} \) threshold)
Iterative Patterns

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>hD; B; A; F; B; A; F; B; C; Ei</td>
</tr>
<tr>
<td>S2</td>
<td>hD; B; A; D; B; B; B; A; B i</td>
</tr>
</tbody>
</table>

Consider the pattern $P = \langle A, B \rangle$

The set of instances of $P$
- $(\text{seq-id, start-pos, end-pos})$
- $\{(1,3,5), (1,6,8), (2,3,5), (2,8,9)\}$
- The support of $P$ is 4
**Frequent Iterative Pattern.** For a trace (sequence) dataset $TDB$, an iterative pattern $P$ is frequent if its instances occur above a certain threshold of $min_{sup}$ in $TDB$.

**Closed Iterative Pattern.** A frequent iterative pattern $P$ is *closed* if there exists no super-sequence $Q$ s.t.:

1. $P$ and $Q$ have the same support;
2. Every instance of $P$ corresponds to a unique instance of $Q$, denoted as $\text{Inst}(P) \approx \text{Inst}(Q)$.

An instance of $P$ $(seq_P, \text{start}_P, \text{end}_P)$ corresponds to an instance of $Q$ $(seq_Q, \text{start}_Q, \text{end}_Q)$ iff $seq_P = seq_Q$ and $\text{start}_P \geq \text{start}_Q$ and $\text{end}_P \leq \text{end}_Q$. 
Closed Unique Iterative Patterns

- |closed patterns| could be too large
  - Due to “noise” in the dataset (e.g., the As in the DB)

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>hA;C;A;A;A;C;A;A;A;A;Ci</td>
</tr>
<tr>
<td>S2</td>
<td>hA;A;A;A;C;A;A;A;A;A;Ci</td>
</tr>
</tbody>
</table>

- At min_sup = 2, patterns \(<A,C>\), \(<A,A,C>\), \(<A,A,A,C>\) and \(<A,A,A,A,C>\) would be reported.
- Due to random interleavings of different noise, number of closed patterns at times is too large.
Closed Unique Pattern. A frequent pattern $P$ is a closed unique pattern if $P$ contains no repeated constituent events, and there exists no super-sequence $Q$ s.t.:

1. $P$ and $Q$ have the same support;
2. Every instance of $P$ corresponds to a unique instance of $Q$;
3. $Q$ contains no constituent events that repeat.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>hA; B; B; B; B; B; C; E; D; A; B; B i</td>
</tr>
<tr>
<td>S2</td>
<td>hC; E; D; A; B; B; B; B; B i</td>
</tr>
</tbody>
</table>

- $<A, B>$ is a closed unique pattern.
- $<C, D>$ is unique but not closed due to $<C, E, D>$
Algorithm 1 Mining Closed Unique Iterative Pattern Growth

Procedure: Mine Closed Unique Pat.
Inputs: $TDB$: Trace database, $min\_sup$: Minimum support
1: Let $FqEv = \{ p | (|p| = 1) \land (sup(p) \geq min\_sup) \}$
2: for every $e$ in $FqEv$
3: Call GrowRec ($e$, $TDB$, $min\_sup$, $FqEv$)

Procedure GrowRec
Inputs: $Pat$: Pattern so far, $TDB$: Trace database, $min\_sup$: Minimum support, $FqEv$: Set of frequent events
4: Let $FqLoc = \{ e \in FqEv | sup(Pat++e) \geq min\_sup \}$
5: if ($Pat$ is closed unique)
6: Output $Pat$
7: if ($Pat$ is unique)
8: for every $f \notin Pat$ in $FqLoc$
9: Let $NPt = Pat++f$
10: if $NPt$ doesn’t satisfy the InfixScan cond. in [LKL-KDD’07]
11: Call GrowRec($NPt$, $TDB$, $min\_sup$, $FqEv$)
Patterns As Features

- Software traces do not come with pre-defined feature vectors
- One could take occurrences of every event as a feature
- However, this would not capture:
  - Contextual relationship
  - Temporal ordering
- We could use mined closed unique patterns as features
Feature Selection

- Select **good features** for classification purpose
- Based on **Fisher score**

\[
F_r = \frac{\sum_{i=1}^{c} n_i (\mu_i - \mu)^2}{\sum_{i=1}^{c} n_i \sigma_i^2}
\]

- \( n_i \) = number of traces in class \( i \) (normal/failure)
- \( \mu_i \) = average feature value in class \( i \)
- \( \sigma_i^2 \) = std. deviation of the feature value in class \( i \)
- the **value** of a feature in a trace/sequence is its num. of instances
Strategy: Select top features so that all traces or sequences are covered at least \( \delta \) times.

---

**Algorithm 2 Feature Selection on Iterative Patterns**

**Procedure: Feature selection**

**Inputs:** \( \mathcal{F} \): Closed Unique Pat., \( TDB \): Trace DB, \( \delta \): Coverage thresh.

**Output:** \( \mathcal{F}_s \): A selected set of iterative patterns

1: Sort iterative patterns in \( \mathcal{F} \) in **decreasing** order of Fisher score;
2: Start with the first pattern \( f_0 \) in \( \mathcal{F} \);
3: **while** (true)
4:   Find the next pattern \( f \);
5:   **if** \( f \) covers at least one sequence in \( TDB \)
6:     \( \mathcal{F}_s = \mathcal{F}_s \cup \{f\} \);
7:     \( \mathcal{F} = \mathcal{F} - \{f\} \);
8: **if** a sequence \( S \) in \( TDB \) is covered \( \delta \) times
9:     \( TDB = TDB - \{S\} \);
10: **if** all sequences are covered \( \delta \) times or \( \mathcal{F} = \phi \)
11:     break;
12: **return** \( \mathcal{F}_s \)
Classifier Building

- Based on the selected discriminative features
- Each trace or sequence is represented as:
  - A feature vector \((x_1, x_2, x_3, \ldots)\)
  - Based on selected iterative patterns
  - The value of \(x_i\) is defined as
    \[
    x_i = \begin{cases} 
    \sup(f_i; S); & \text{if } S \text{ contains } f_i \\
    0; & \text{otherwise}
    \end{cases}
    \]
- Train an \textbf{SVM} model
  - Based on two contrasting sets of feature vectors
Synthetic Datasets

- Trace generators QUARK [LK-WCRE’06]
- Input software models with injected errors
- Output a set of traces with labels

Real traces (benchmark programs)

- Siemens dataset (4 largest programs)
- Used for test-adequacy study - large number of test cases, with injected bugs, correct output available
- Inject multiple bugs, collect labeled traces

Real traces (real program, real bug)

- MySQL dataset
- datarace bug
Experiments: Eval. Details

- Performance measures used
  - Classification accuracy
  - Area under ROC curve

- 5 Fold-Cross Validation
  - Mining, feature selection and model building done for each fold separately
  - Prevent information leak

- Handling skewed distribution
  - Failure training data is duplicated many times
  - Test set distribution is retained

- Three types of bugs
  - Addition, omission and ordering
### Experimental Results: Synthetic

#### Data Set Accuracy

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Correct (jtracesj)</th>
<th>Error (jtracesj)</th>
<th>Add/Omis.</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>X11</td>
<td>125</td>
<td>125</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CVS Omission</td>
<td>170</td>
<td>170</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CVS Ordering</td>
<td>180</td>
<td>0</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>CVS Mix</td>
<td>180</td>
<td>90</td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>

#### Data Set Accuracy

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Evt</th>
<th>Pat</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Evt</td>
<td>Pat</td>
<td>Evt</td>
</tr>
<tr>
<td>X11</td>
<td>96:40 § 4:10</td>
<td>97.20 § 3.35</td>
<td>0:97 § 0:04</td>
</tr>
<tr>
<td>CVS Omission</td>
<td>95:29 § 1:61</td>
<td>100.00 § 0.00</td>
<td>0:96 § 0:03</td>
</tr>
<tr>
<td>CVS Ordering</td>
<td>50:00 § 0:00</td>
<td>85.28 § 2.71</td>
<td>0:50 § 0:00</td>
</tr>
<tr>
<td>CVS Mix</td>
<td>66:39 § 15:63</td>
<td>93.89 § 5.94</td>
<td>0:65 § 0:17</td>
</tr>
</tbody>
</table>
Experimental Results: Siemens & MySQL

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Correct (jtracesj)</th>
<th>Error (jtracesj)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tot_info</td>
<td>schedule</td>
</tr>
<tr>
<td></td>
<td>302</td>
<td>2140</td>
</tr>
<tr>
<td></td>
<td>208</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>1851</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Accuracy</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Evt</td>
<td>Pat</td>
</tr>
<tr>
<td></td>
<td>tot_info</td>
<td>schedule</td>
</tr>
<tr>
<td></td>
<td>77:33 § 2:31</td>
<td>52:83 § 19:27</td>
</tr>
<tr>
<td></td>
<td>90.67 § 5.82</td>
<td>86.26 § 14.90</td>
</tr>
<tr>
<td></td>
<td>0:90 § 0:03</td>
<td>0:57 § 0:25</td>
</tr>
<tr>
<td></td>
<td>0.94 § 0.03</td>
<td>0.88 § 0.16</td>
</tr>
</tbody>
</table>
### Experimental Results: Varying Min-Sup

<table>
<thead>
<tr>
<th>min_sup</th>
<th>Accuracy</th>
<th>A U C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>90:9497 § 2:9203</td>
<td>0:9344 § 0:0454</td>
</tr>
<tr>
<td>0.10</td>
<td>90:9497 § 2:9203</td>
<td>0:9344 § 0:0454</td>
</tr>
<tr>
<td>0.15</td>
<td>90:9004 § 2:5949</td>
<td>0:9323 § 0:0509</td>
</tr>
<tr>
<td>0.20</td>
<td>90:8939 § 2:5949</td>
<td>0:9321 § 0:0499</td>
</tr>
<tr>
<td>0.25</td>
<td>90:8380 § 2:5402</td>
<td>0:9318 § 0:0506</td>
</tr>
<tr>
<td>0.30</td>
<td>90:7263 § 2:5555</td>
<td>0:9310 § 0:0501</td>
</tr>
<tr>
<td>0.35</td>
<td>90:2794 § 2:8650</td>
<td>0:9261 § 0:0545</td>
</tr>
<tr>
<td>0.40</td>
<td>90:2794 § 2:8650</td>
<td>0:9261 § 0:0545</td>
</tr>
<tr>
<td>0.45</td>
<td>90:2794 § 2:8650</td>
<td>0:9261 § 0:0545</td>
</tr>
<tr>
<td>0.50</td>
<td>90:2794 § 2:8650</td>
<td>0:9261 § 0:0545</td>
</tr>
</tbody>
</table>

Replace dataset
Experimental Results: Mining Time

Replace dataset

**Mining Closed Unique Iterative Patterns**

**Mining Closed Patterns: Cannot run at support 100%**

(Out of memory exception, 1.7GB memory, 4 hours)
- Pattern-based classification
  - Itemsets: Cheng et al. [ICDE’07, ICDE’08]
  - Graphs: Yan et al. [SIGMOD’08]
- Mining episodes
  - Mannila et al. [DMKD’97]
- Mining repetitive sub-sequences
  - Ding et al. [ICDE’09]
- Dickinson et al. [ICSE’01]
  - Clustering program behaviors
  - Detection of failures by looking for small clusters
- Bowring et al. [ISSTA’04]
  - Model failing trace and correct trace as first order Markov model to detect failures
New pattern-based classification approach
- Working on repetitive sequential data
- Applied for failure detection

Classification accuracy improved by 24.68%
- Experiments on different datasets
- Different bug types: omission, addition, ordering

Future work
- Direct mining of discriminative iterative patterns
- Application of the classifier to other form of sequential data:
  - Textual data, genomic & protein data
  - Historical data
- Pipelining to SE tools: fault localization tools, test suite augmentation tools
Thank You

Questions, Comments, Advice?