SMU
SINGAPORE MANAGEMENT UNIVERSITY
School of Information Systems

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- Existing problems in specification: Lack, incomplete and outdated specifications [LK06,ABL02,YEBBD06, DSB04, etc.]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

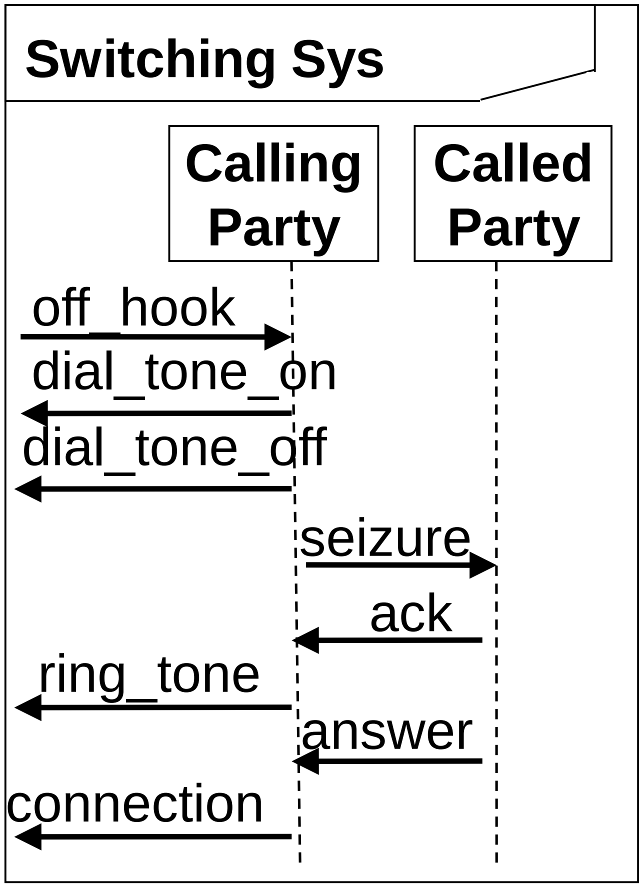
- Episode Mining [MTV97,G03] - A series of closely-occurring events recurring frequently within a sequence

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



[ITU]

- TS1: off_hook, seizure, ack, ring_tone, answer, ring_tone, connection_on ✗
- TS2: off_hook, seizure, ack, ring_tone, answer, answer, answer, connection_on ✗
- TS3: off_hook, seizure, ack, ev1, ring_tone, ev1, answer, connection_on ✓

Iterative Patterns - Semantics

- Given a pattern P $(e_1e_2\ldots e_n)$, a substring SB is an instance of P iff

  $$SB = e_1;[-e_1,\ldots,e_n]^*;e_2;\ldots;[-e_1,\ldots,e_n]^*;e_n$$

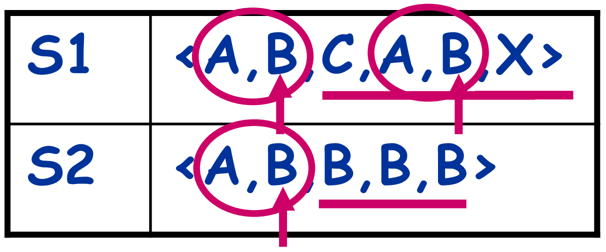
- Pattern: *<off_hook, seizure, ring_tone, answer, connection_on>*
- S1: off_hook, ring_tone, seizure, answer, connection_on ✗
- S2: off_hook, seizure, ring_tone, answer, answer, answer, connection_on ✗
- S3: off_hook, seizure, ev1, ring_tone, ev1, answer, connection_on ✓
- S4: |off_hook, seizure, ev1, 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 – $SeqDB_P^{all}$
  Return: All suffixes of sequences in SeqDB where for each, its infix is an instance of pattern P

$SeqDB$



S1	<A,B,C,A,B,X>
S2	<A,B,B,B,B>

$SeqDB_{\langle A,B \rangle}^{all}$

(Seq,Start,End)	Sequence
(1,1,2)	<C,A,B,X>
(1,4,5)	<X>
(2,1,2)	<B,B,B>

- Support of a pattern = size of its proj. DB
- $SeqDB_{ev}^{all}$ is formed by considering occurrences of ev
- $SeqDB_{P{+}{+}ev}^{all}$ can be formed from $SeqDB_P^{all}$

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

S1	<X,A,B,B,C,D>
S2	<X,A,B,B,C,D,E,F,G >
S3	<B,C,A,D,E,D>

Pattern: <A,C>

Prefix Ext: {<X>}

Suffix Ext: {<D>}

Infix Ext: {<B>}

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 $\notin SeqDB_P^{all}$, then we can stop growing P.

S1	<X,A,B,B,C,D>
S2	<X,A,B,B,C,D,E,F,G>
S3	<B,C,A,D,E,D>

Pattern: <A,C>

Prefix Ext: {<X>}

Infix Ext: {<B>}

Suffix Ext: {<D>}

<A,C> is not closed and we can stop growing it.
No need to check for <A,C,…>

```
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 ($f\_ev$, $SeqDB^{all}_{f\_ev}$, $min\_sup$, $Closed$, $Freq$)
```

Main Method

```
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 $\{e|e \in EV \wedge (\sup(Pat{+\!\!+}e) \geq min\_sup)$
7:     Let NxtPat = $Pat{+\!\!+}f\_ev$
8:     If ($\nexists\ e.\ (e \in$ InfixExt(NxtPat) $\wedge\ e \notin SeqDB^{all}_{NxtPat}$))
9:       Call MineRecurse (NxtPat, ProjDB, $min\_sup$, $Closed$, $EV$)
```

Recursive Pattern Growth

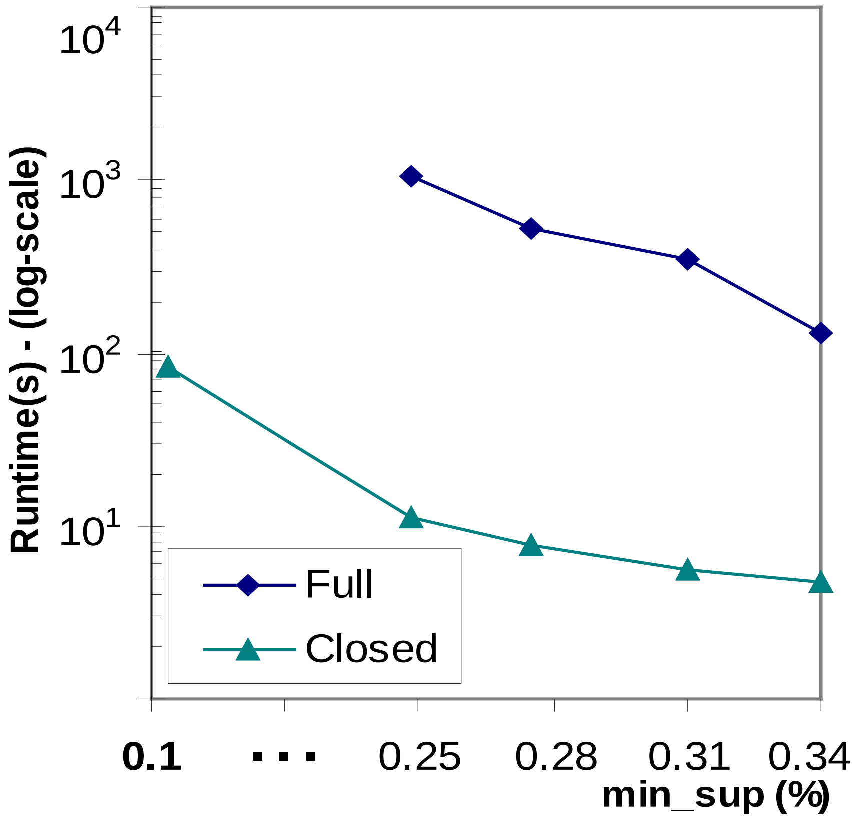
Closure Checks

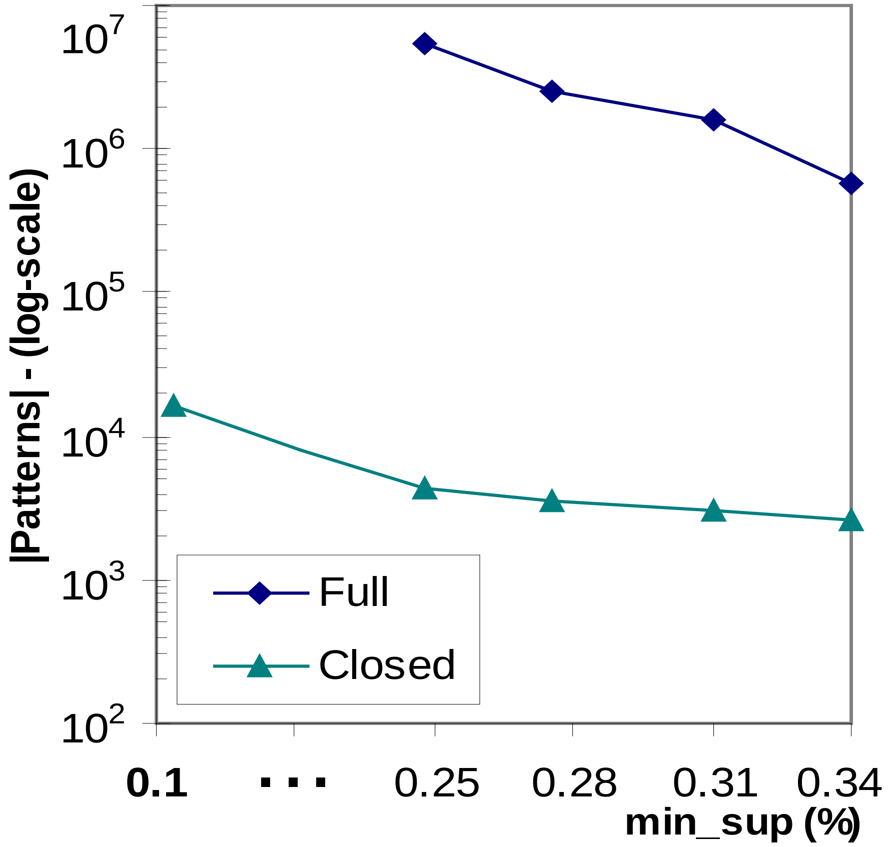
InfixScan Pruning

Performance & Case Studies

Performance Study - I

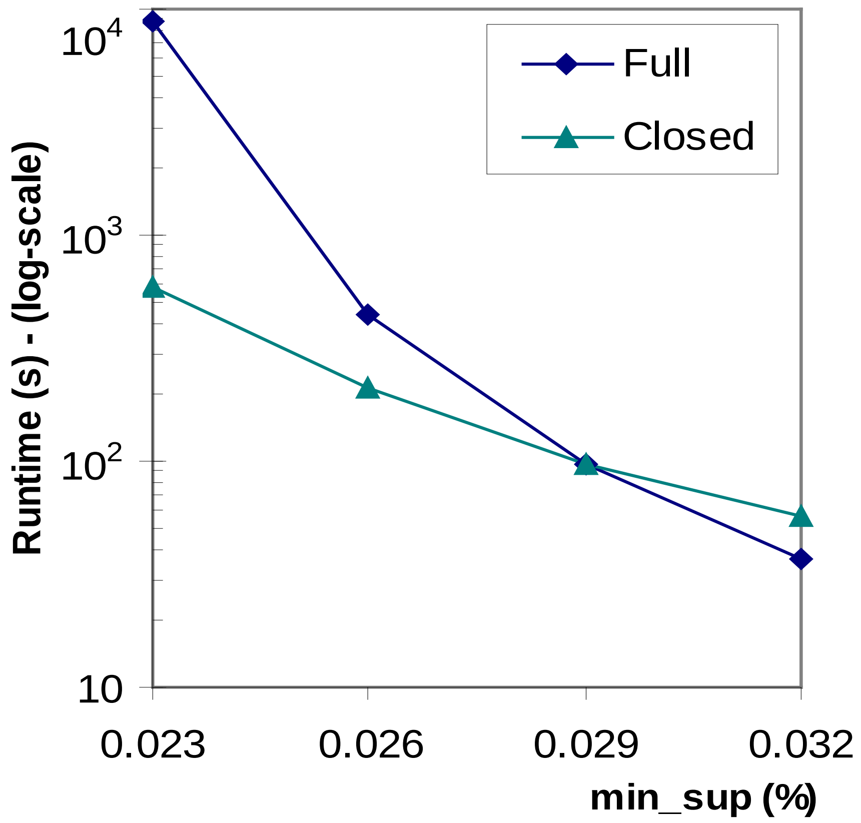
- **Synthetic Dataset**
  - IBM Simulator : D5C20N10S20

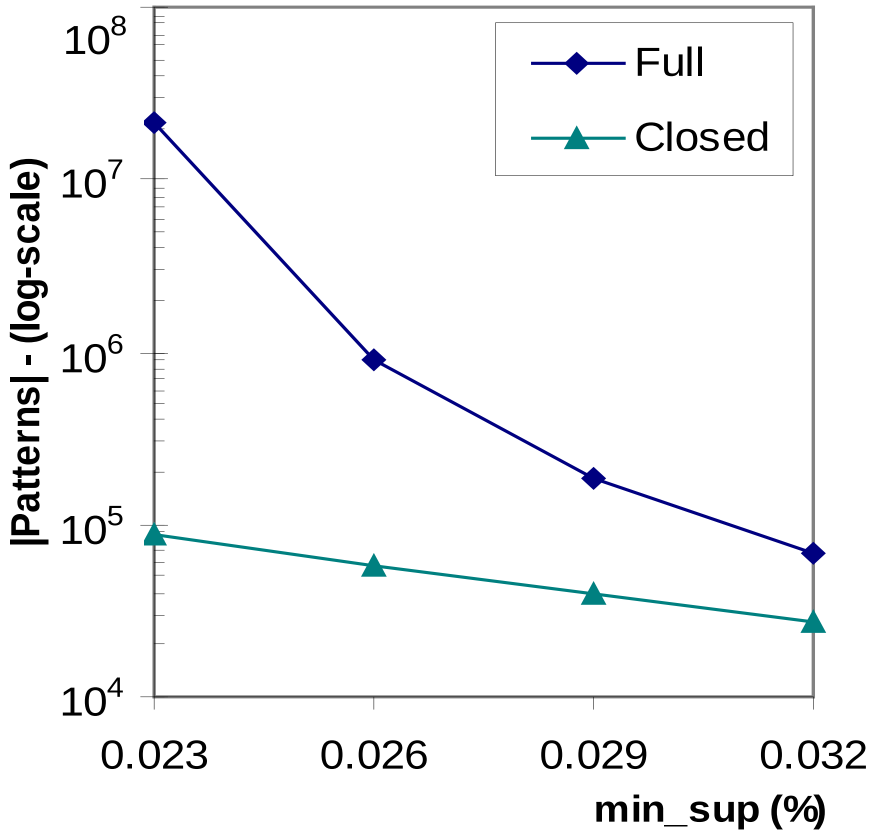




Performance Study - II

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

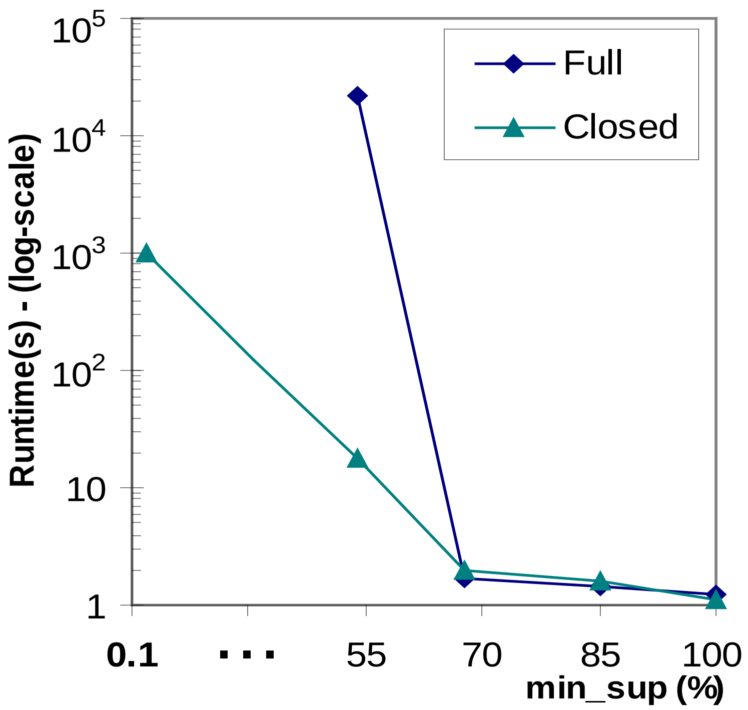


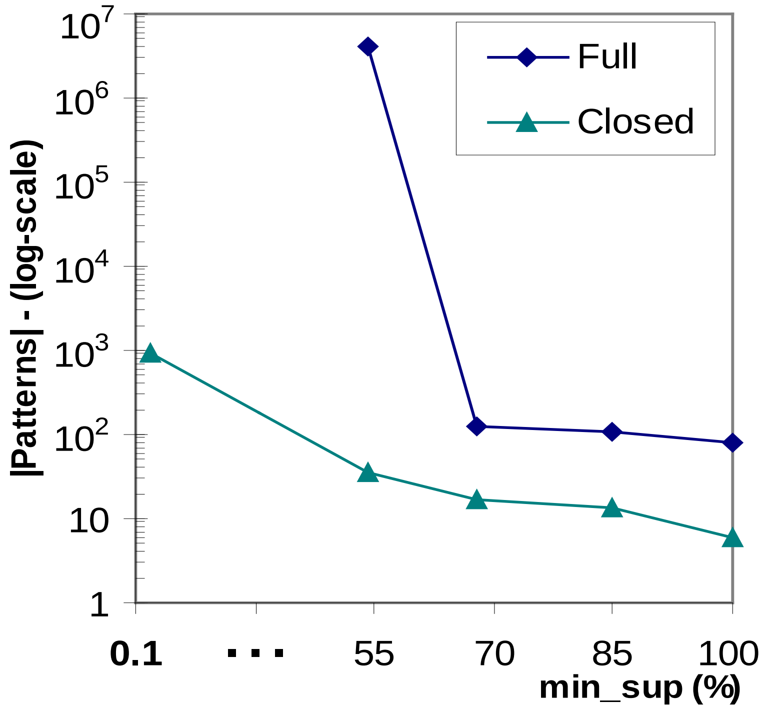


Performance Study - III

- **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	TransactionManagerLocator.getInstance TransactionManagerLocator.locate TransactionManagerLocator.tryJNDI TransactionManagerLocator.usePrivateAPI	**Transaction Commit**	TxManager.commit TransactionImpl.commit TransactionImpl.beforePrepare TransactionImpl.checkIntegrity TransactionImpl.checkBeforeStatus TransactionImpl.endResources TransactionImpl.completeTransaction TransactionImpl.cancelTimeout TransactionImpl.doAfterCompletion TransactionImpl.instanceDone
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 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(,,,);

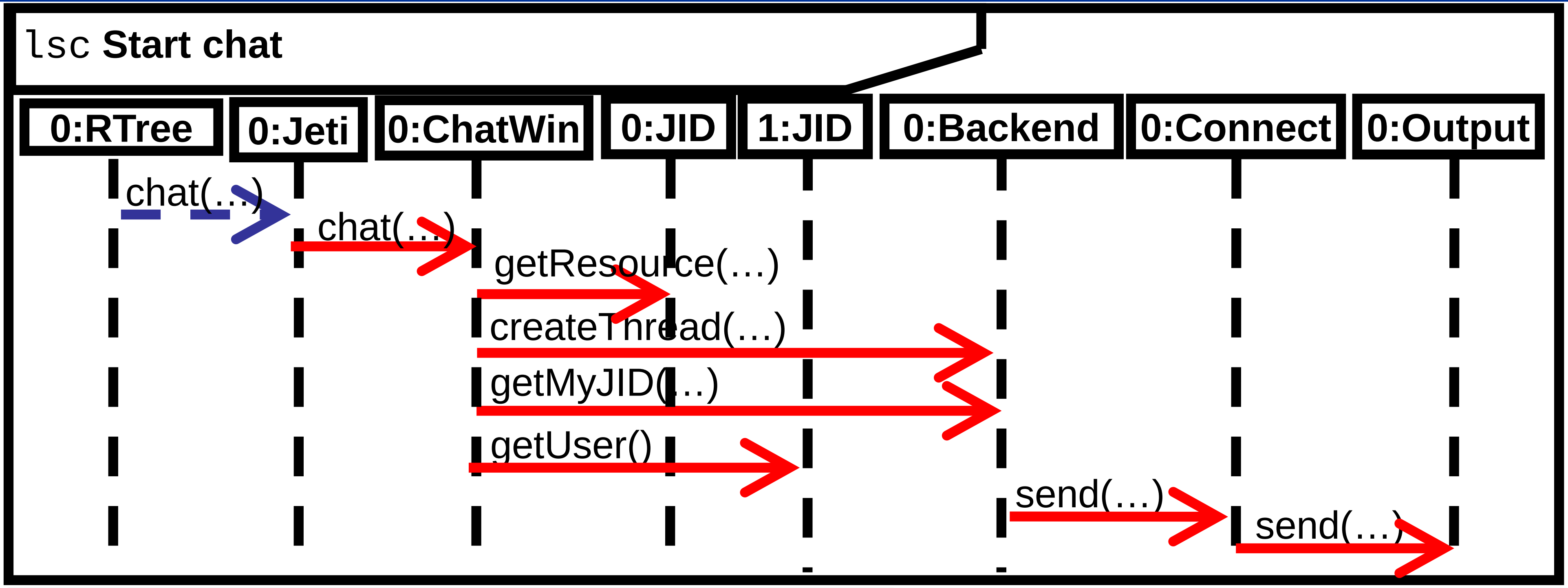
Detect double free, which is disallowed
"Calling HeapFree twice with the same pointer can cause heap corruption, resulting in subsequent calls to HeapAlloc returning the same pointer twice." [MSDN]

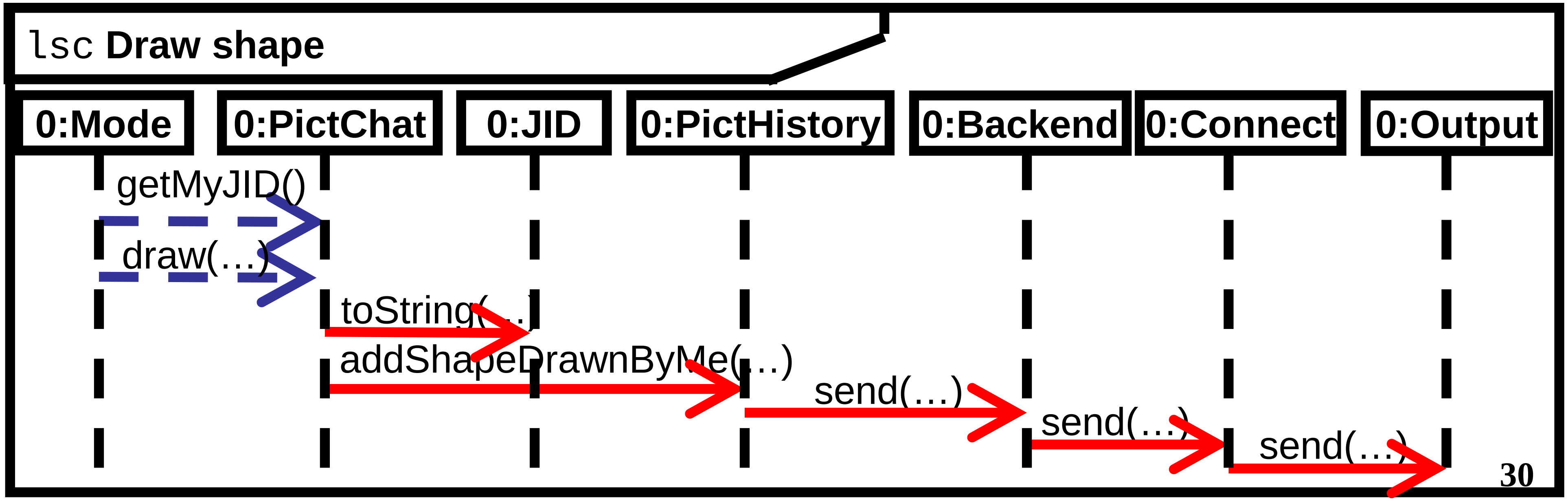
Library Usage & Bug Detection: Windows Application -- Extension

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

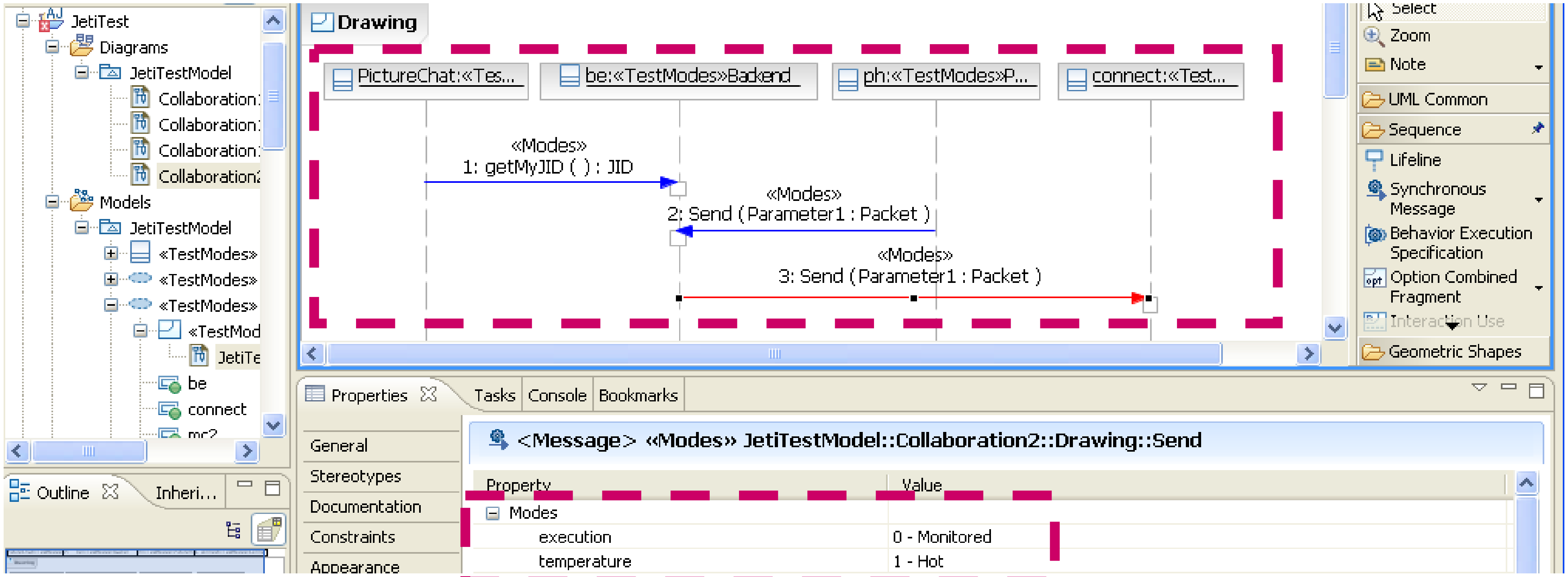
V CreateCompatDC(); -> DeleteDC(V);
V CreCompatBmap(,,);->DeleteObj (V);
V CreRectRgn(,,,)-> DeleteObj(V);
DeleteDC(V) –precede-> V CreCompDC()
SetBkColor(,V); -> V SetBkColor (,)
…

Mined LSCs -Jeti Mess. App





LSC Visualization & Scenario-Based Test



<Message> «Modes» JetiTestModel::Collaboration2::Drawing::Send

Property	Value
Modes	
execution	0 - Monitored
temperature	1 - Hot

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 Lo[1], Hong Cheng[2], Jiawei Han[3], Siau-Cheng Khoo[4], and Chengnian Sun[4]

[1]Singapore Management University, [2]Chinese University of Hong Kong, [3]University of Illinois at Urbana-Champaign, [4]National 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 ?

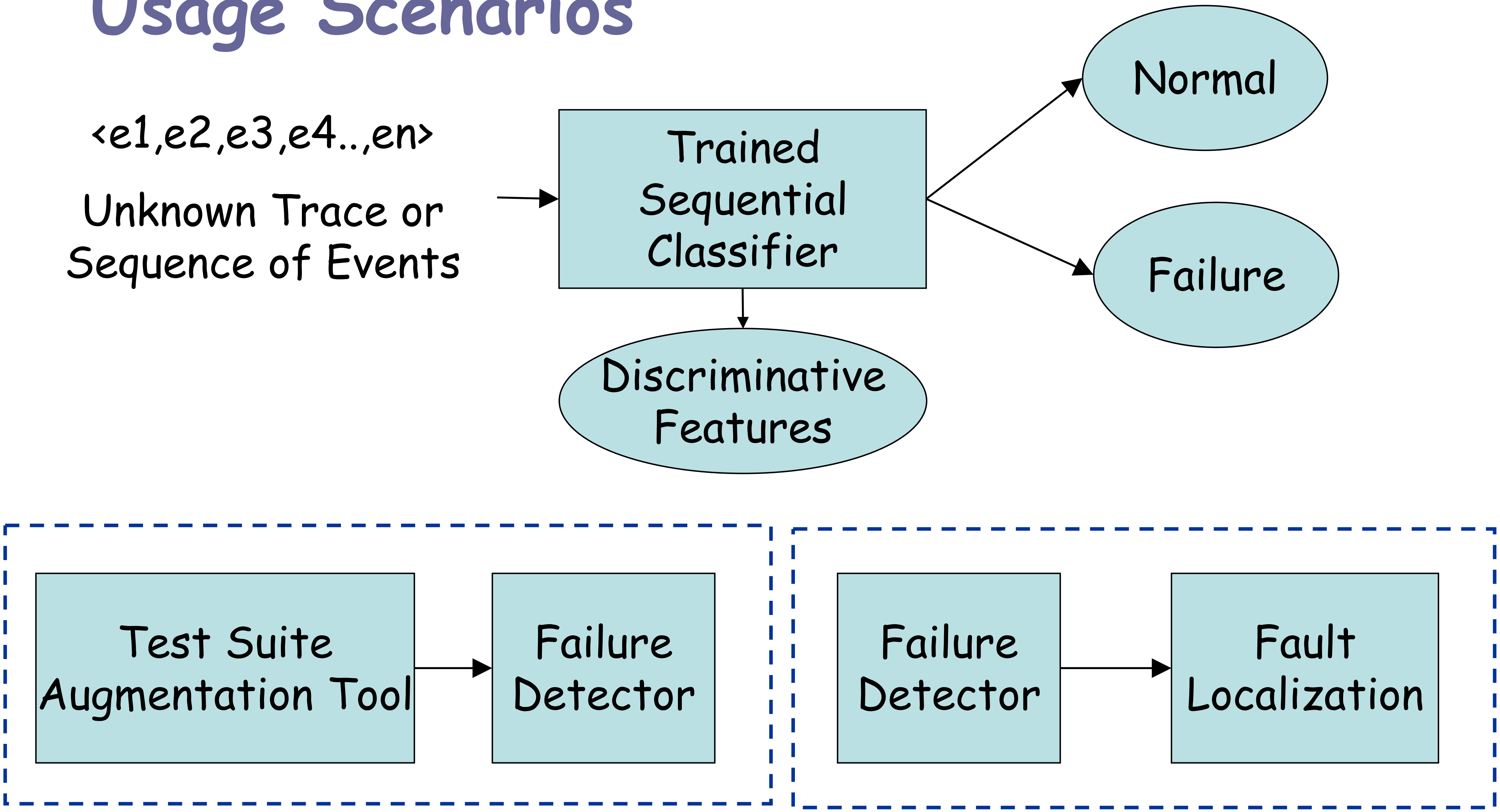
- Pattern mining tool for program behaviors
- Recent development of the pattern-based classification approach
- In this work, we extend the above work to:
  - Propose a new pattern definition which could be more efficiently mined (closed unique iterative pattern)
  - Develop a new pattern-based classification 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



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)

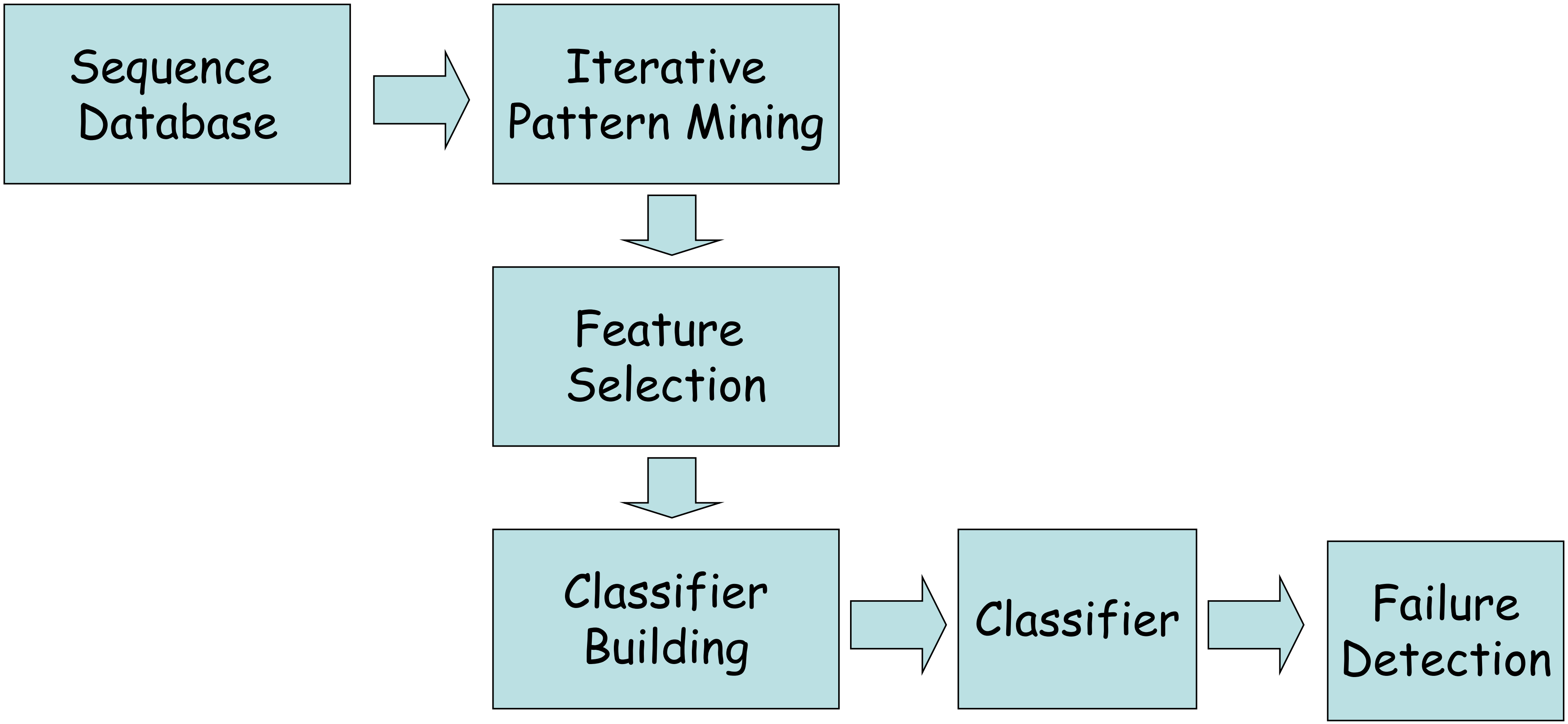
Research Questions

- 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 $<e_1,e_2,e_3,\ldots,e_n>$
- An event, is a unit behavior of interest
  - Method call
  - Statement execution
  - Basic block execution in a Control Flow Graph (CFG)
- Input traces -> a sequence database

Overall View of The Pattern-Based Classification Framework



Iterative Patterns

- A pattern is a series of events (P=<p_1,p_2,..,p_n>)
- 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 ($e_1e_2…e_n$), a substring SB is an instance of P iff

  SB = e_1;[-e_1,…,e_n]*;e_2;…;[-e_1,…,e_n]*;e_n
- Goal: Find patterns whose instances appear often within a sequence and across multiple sequences (above a min_sup threshold)

Iterative Patterns

Identi¯er	Sequence
S1	hD ; B ; A ; F ; B ; A ; F ; B ; C ; E i
S2	hD ; B ; A ; D ; B ; B ; B ; A ; B i

- Consider the pattern P = <A,B>
- The set of instances of P
  - (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 vs. Closed Patterns

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 $\mathrm{Inst}(P) \approx \mathrm{Inst}(Q)$.

An instance of P $(seq_P, start_P, end_P)$ corresponds to an instance of Q $(seq_Q, start_Q, end_Q)$ iff $seq_P = seq_Q$ and $start_P \geq start_Q$ and $end_P \leq end_Q$.

Closed Unique Iterative Patterns

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

Identi¯er	Sequence
S1	hA; C; A; A; A; C; A; A; A; Ci
S2	hA; A; A; A; C; A; A; A; A; Ci

- 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.

Identi¯er	Sequence
S1	hA ; B ; B ; B ; B ; C ; E ; D ; A ; B ; B i
S2	hC ; E ; D ; A ; B ; B ; B ; B ; B i

- <A,B> is a closed unique pattern.
- <C,D> is unique but not closed due to <C,E,D>

Mining Algorithm

```
Algorithm 1 Mining Closed Unique Iterative Pa
Procedure: Mine Closed Unique Pat.
Inputs: $TDB$: Trace database, $min\_sup$: Minimum support
1: Let $FqEv = \{p | (|p| = 1) \wedge (sup(p) \geq min\_sup)\}$
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($NPt$, $TDB$, $min\_sup$, $FqEv$)
```

Main Method

Recursive Pattern Growth

Closure & Uniqueness Checks

Pruning

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

$$Fr = \frac{\sum_{i=1}^{c} n_i (\mu_i - \mu)^2}{\sum_{i=1}^{c} n_i \sigma_i^2}$$

- n_i = number of traces in class i (normal/failure)
- μ_i = average feature value in class i
- σ_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

Feature Selection

- Strategy: Select **top features** so that all traces or sequences are **covered at least δtimes**.

```
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 SVM model
  - Based on two contrasting sets of feature vectors

Experiment: Datasets

- 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

Dataset	Correct	Error (jtracesj)	
	(jtracesj)	Add/Omis.	Order
X11	125	125	0
CVS Omission	170	170	0
CVS Ordering	180	0	180
CVS Mix	180	90	90

Dataset	Accuracy		AUC	
	Evt	Pat	Evt	Pat
X11	96:40 § 4:10	97.20 § 3.35	0:97 § 0:04	1.00 § 0.00
CVS Omission	95:29 § 1:61	100.00 § 0.00	0:96 § 0:03	1.00 § 0.00
CVS Ordering	50:00 § 0:00	85.28 § 2.71	0:50 § 0:00	0.82 § 0.08
CVS Mix	66:39 § 15:63	93.89 § 5.94	0:65 § 0:17	0.95 § 0.06

Experimental Results: Siemens & MySQL

Dataset	Correct	Error (jtracesj)	
	(jtracesj)	Add/Omis	Order
tot_info	302	208	94
schedule	2140	289	1851
print_tokens	3108	187	187
replace	1259	269	269
MySQL	51	0	51

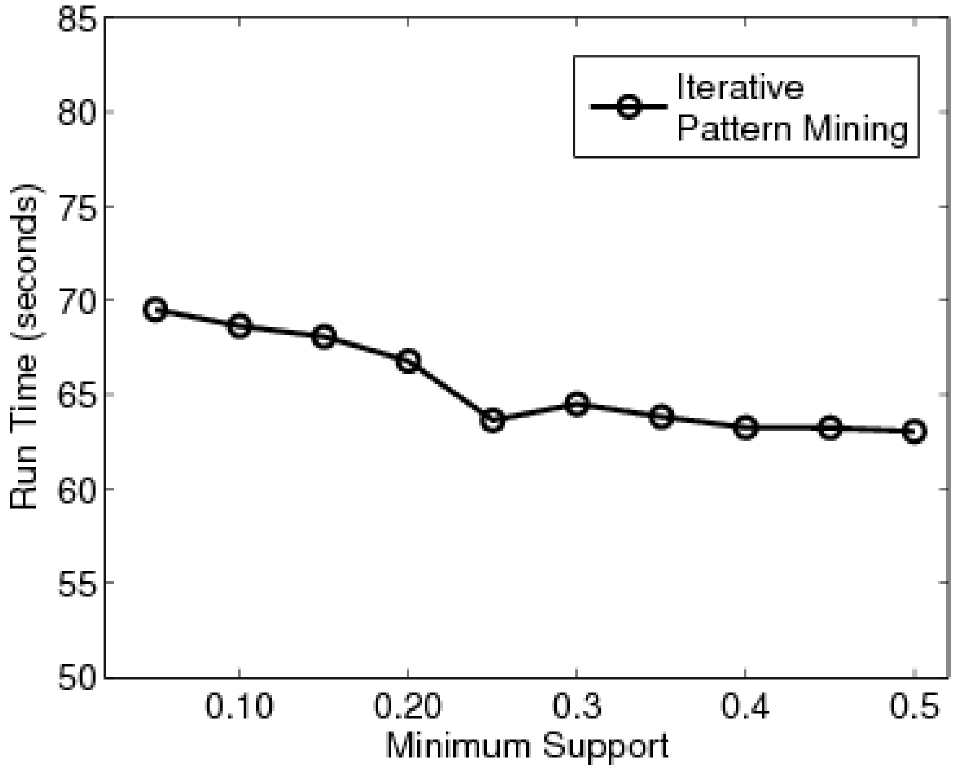
Dataset	Accuracy		AUC	
	Evt	Pat	Evt	Pat
tot_info	77:33 § 2:31	90.67 § 5.82	0:90 § 0:03	0.94 § 0.03
schedule	52:83 § 19:27	86.26 § 14.90	0:57 § 0:25	0.88 § 0.16
print_tokens	72:60 § 26:33	99.94 § 0.08	0:64 § 0:17	1.00 § 0.00
replace	61:12 § 9:25	90.84 § 2.54	0:63 § 0:15	0.93 § 0.05
MySQL	50:00 § 0:00	100.00 § 0.00	0:50 § 0:00	1.00 § 0.00

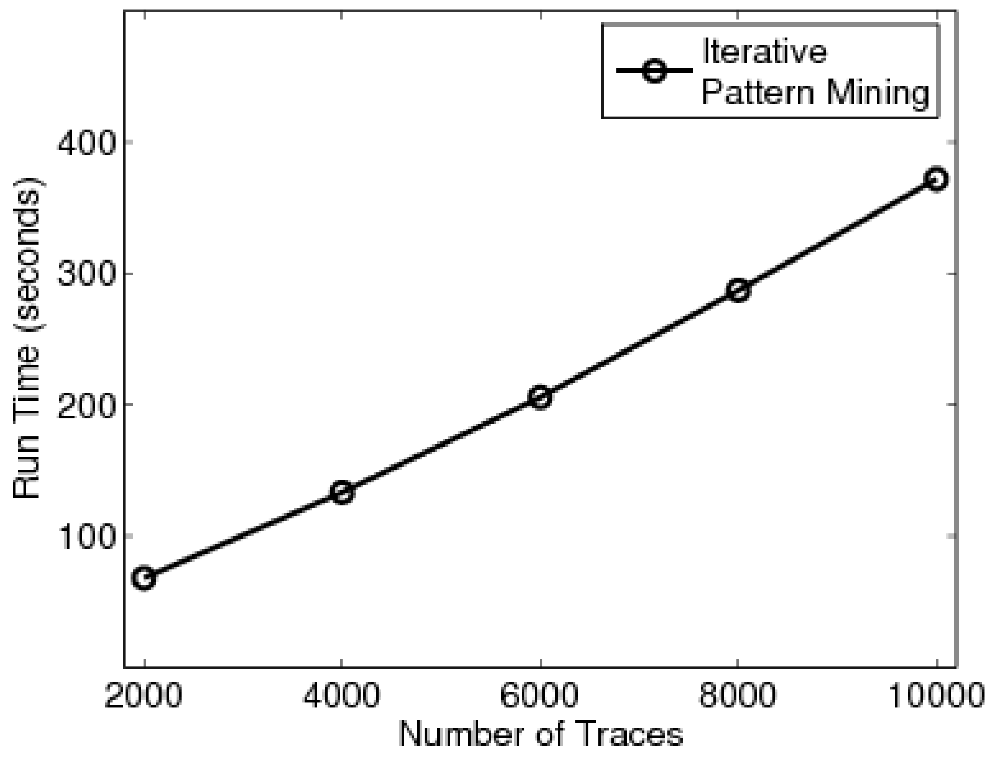
Experimental Results: Varying Min-Sup

min_sup	Accuracy	AUC
0.05	90:9497 § 2:9203	0:9344 § 0:0454
0.10	90:9497 § 2:9203	0:9344 § 0:0454
0.15	90:9004 § 2:5949	0:9323 § 0:0509
0.20	90:8939 § 2:5949	0:9321 § 0:0499
0.25	90:8380 § 2:5402	0:9318 § 0:0506
0.30	90:7263 § 2:5555	0:9310 § 0:0501
0.35	90:2794 § 2:8650	0:9261 § 0:0545
0.40	90:2794 § 2:8650	0:9261 § 0:0545
0.45	90:2794 § 2:8650	0:9261 § 0:0545
0.50	90:2794 § 2:8650	0:9261 § 0:0545

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)

Related Work

- 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

Conclusion & Future Work

- 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 ?