Fouille de motifs basée sur la programmation par contraintes
- Appliquée à la validation de logiciels -

SOUTENANCE DE THÈSE DE DOCTORAT LMD

Mehdi MAAMAR

Université d'Oran 1, Laboratoire LITIO, Équipe PCO

Laboratoire LIRMM, Équipe Coconut

Oran, 29 Mai 2017

Directeur : *Yahia LEBBAH*

Co-directeur : *Nadjib LAZAAR*

Outline

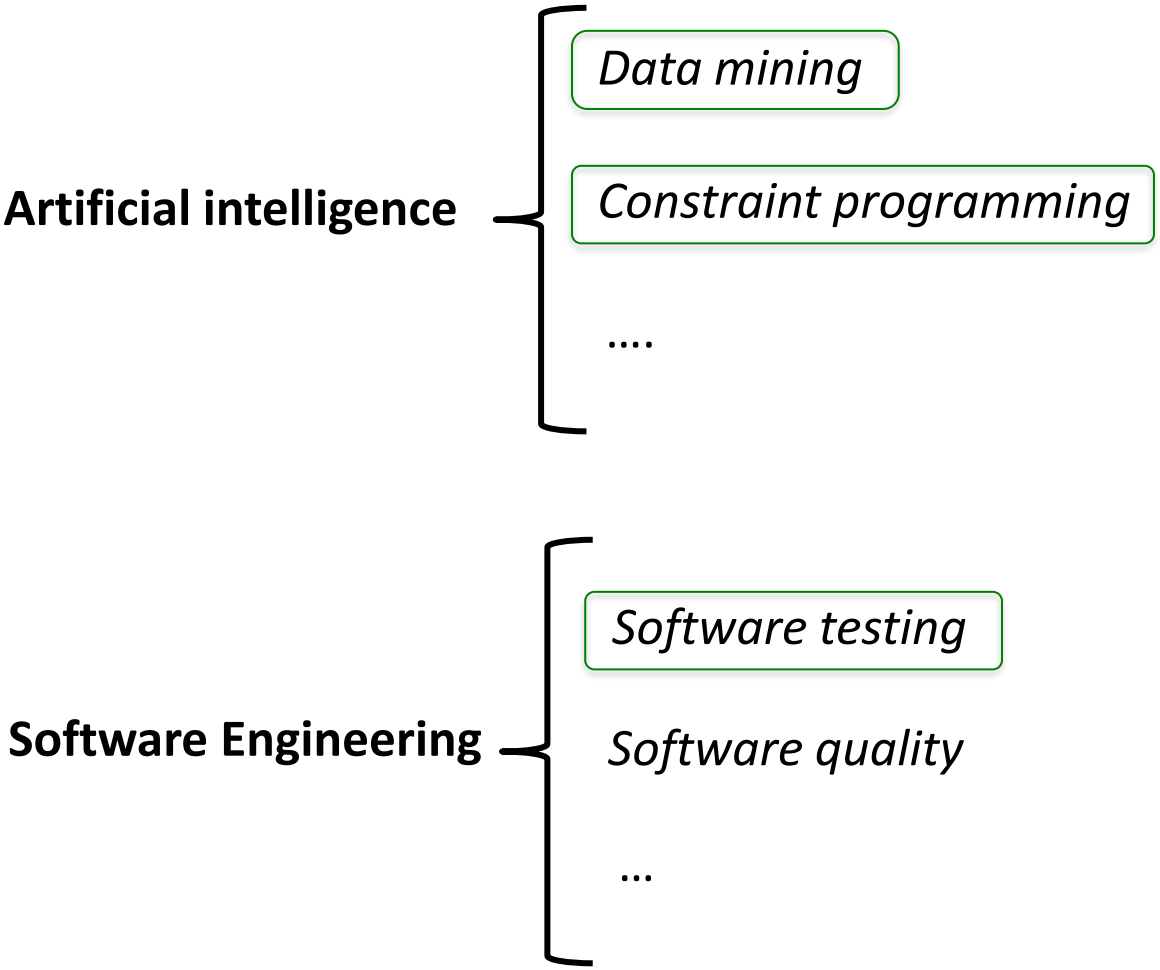
- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* Itemset mining for FL
- *Contribution 3* : *F-CPMiner* Tool
- Conclusions & perspectives

Outline

- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* itemset mining for FL
- *F-CPMiner Tool :* implementation & results
- Conclusions & perspectives

Context & Motivations

The thesis concerns :

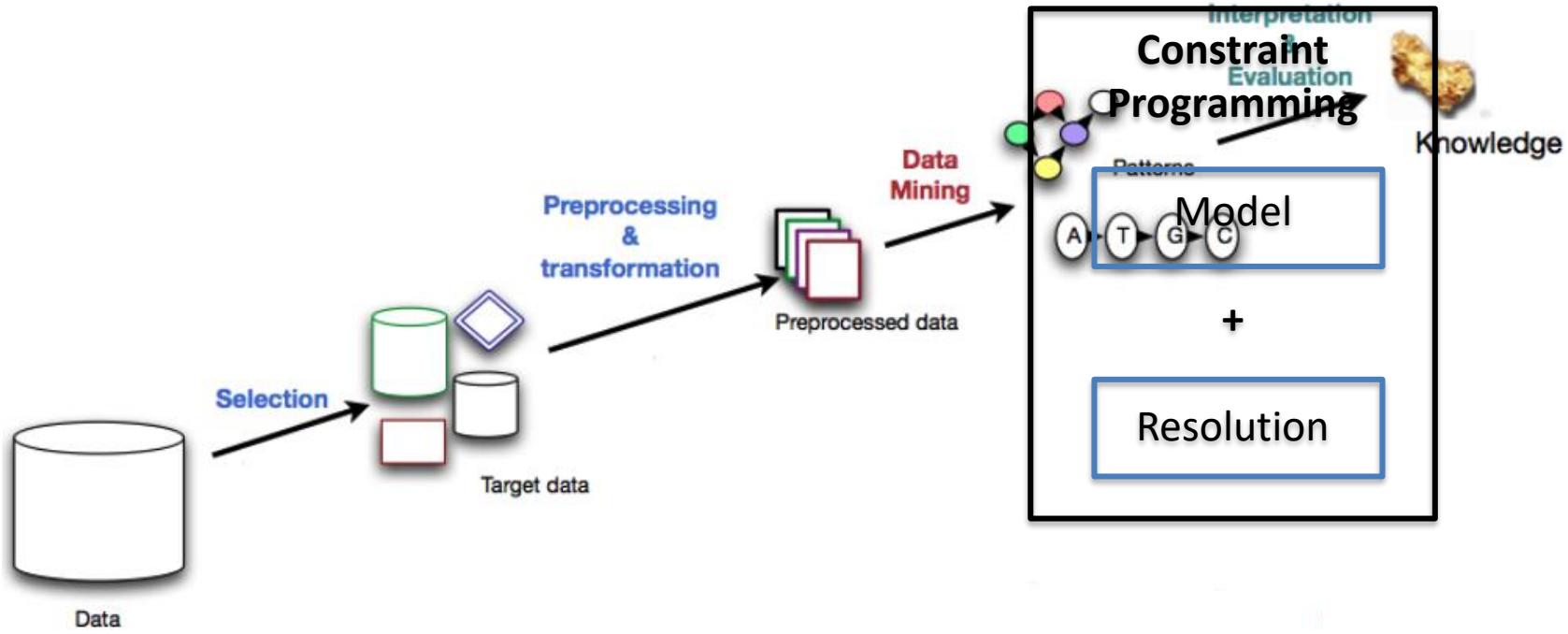


Context & Motivations

Itemset mining
- Complex queries

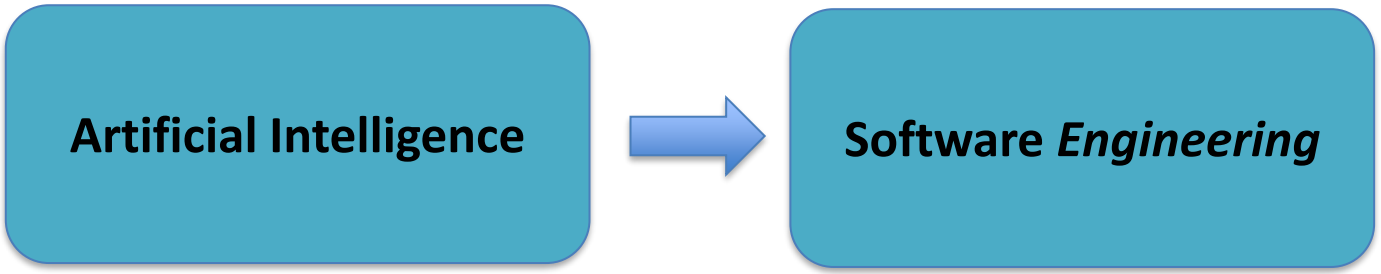
+

Constraint programming
- Declarative paradigm



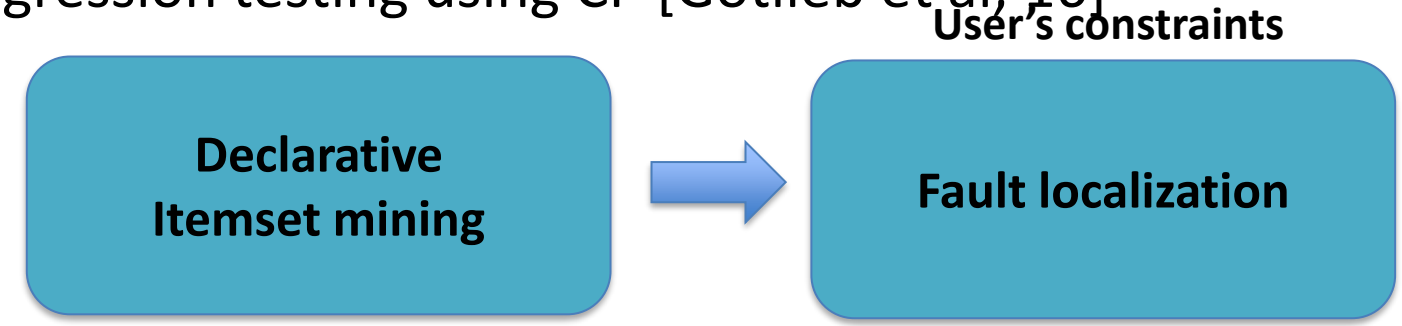
Context & Motivation

Application domains (IA & SE):



- Neural network in the software testing [vanmali et al, 02]
- Regression testing using CP [Gotlieb et al, 16]

...



***RQ* 2: How declarative IM can improve fault localization task?**

Context & Motivation

Research Questions :

***RQ* 1:** **Can CP be more efficient for itemset mining ?**

→ **Contribution 1**

***RQ* 2:** **How declarative IM can improve Fault Localization task?**

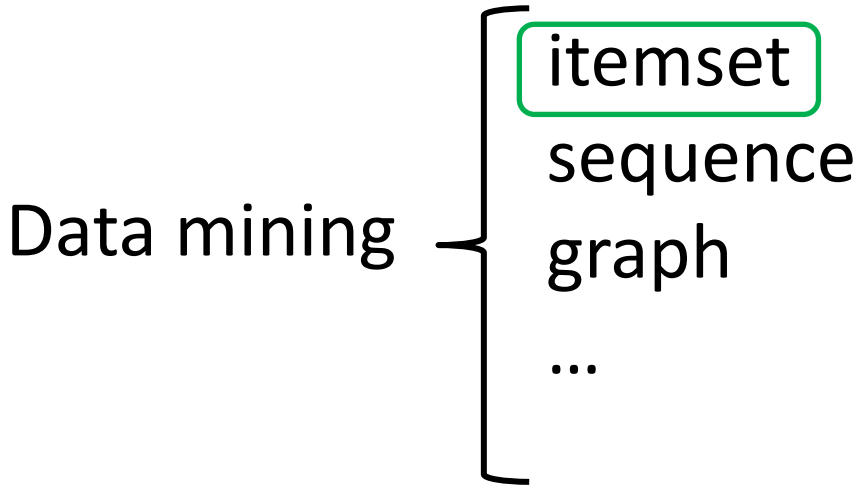
→ **Contribution 2**

Outline

- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* IM for FL
- *F-CPMiner Tool :* implementation & results
- Conclusions & perspectives

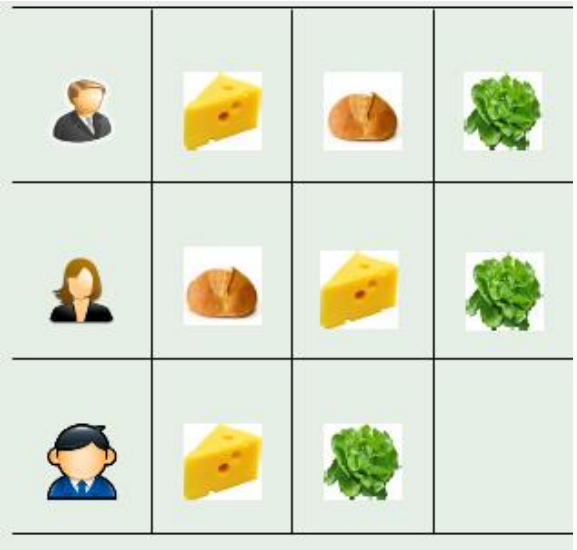
Itemset Mining : Definition

- A well-known and the most popular research field of data mining [Agrawal, 93]



- It aims at finding regularities in a dataset

Itemset Mining: Super Market example



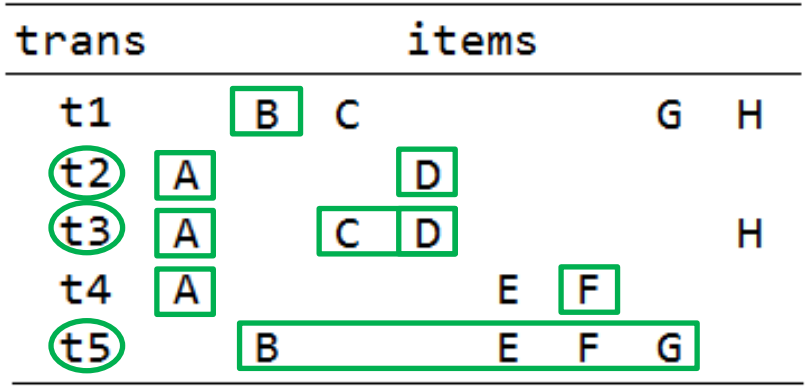
The set of items I = 

The set of transactions T = 

Find sets of products that are frequently bought together

Aim : Optimize the organization of the offered products

Itemset Mining: Example 2



trans	items
t1	B C G H
t2	A D
t3	A C D H
t4	A E F
t5	B E F G

Set of items: I = {A,B,C,D,E,F,G,H}

Set of transactions: T = {t1,t2,t3,t4,t5}

Itemset: P ⊆ I **Language :** $L_I = 2^{|I|}$

Cover(P) = {t ∈ T | P ⊆ t}

Cover(AD)= {t2,t3}

Cover(BEFG)= {t5}

Frequent Itemset Mining

- The frequency of an itemset is the size of its cover

$$freq(P) = |cover(P)|$$

- Let $\theta \in \mathbb{N}^+$ be the minimum support

Frequent itemset mining problem :

- Extract all itemsets P satisfying : $freq(P) \geq \theta$

trans	items
t1	B C E G H
t2	A D E F
t3	A C D H
t4	A D E F
t5	B E F G

$freq(BC) = 1$

Frequent itemset with $\theta = 3$

$freq(EG) = 2$

A:3
D:3
E:4
F:3
AD:3
EF:3

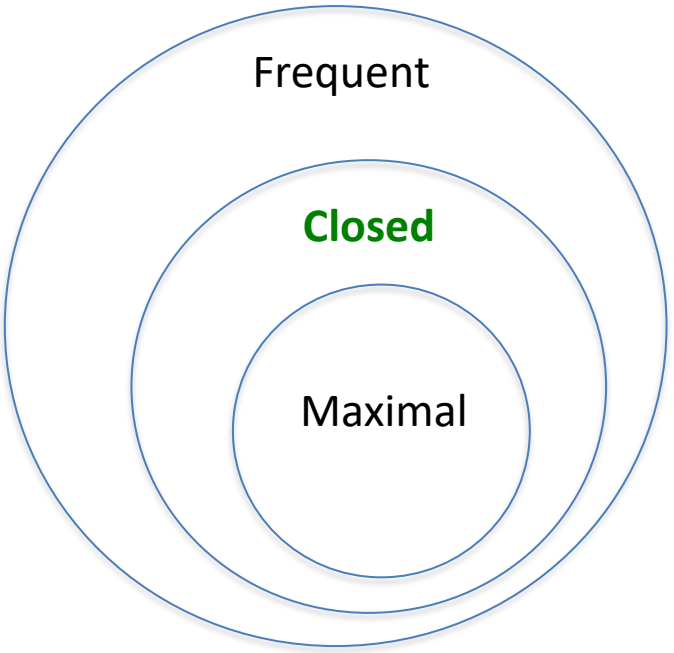
Can we compact these itemsets ?

Condensed representations [Pasquier, 99]

Most of frequent itemsets are redundant :
possible to derive them from other patterns

Maximal patterns : reconstruct the set of frequent patterns

Closed patterns : reconstruct the set of all frequent patterns and their frequencies



$M \subseteq C \subseteq F$

Closed itemsets : Definition

- P is a closed itemset :

$$\forall P \subset Q : freq(Q) < freq(P)$$

trans	items						
t1		B	C		E		G	H
t2	A			D	E	F		
t3	A		C	D				H
t4	A			D	E	F		
t5		B			E	F	G	

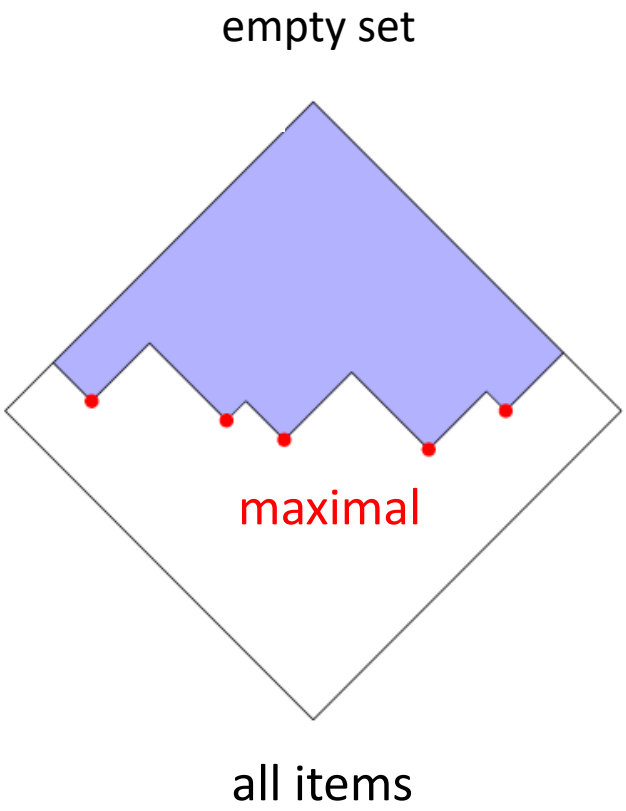
Frequent itemset with $\theta = 3$

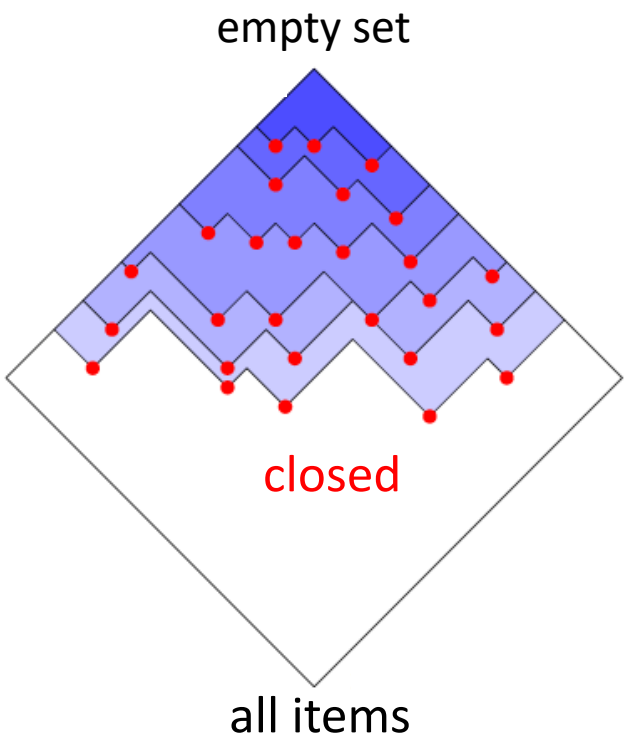
A:3
D:3
E:4
F:3
AD:3
EF:3

are not closed

are closed

Condensed representations : Lattice





Closed Frequent Itemset Mining

The Need:

Extract **all** Closed Frequent Itemsets

How?

Closed Frequent Itemset: State of art

Specialized algorithms [Pasquier et al. 99, Uno et al. 03, ..]

→ **Many efficient algorithms for mining closed itemsets** ✓

(*CLOSE*, *FP-Growth*, *LCM*…)

But

- Dedicated to particular classes of constraints

  ↳ very hard to combine different constraints

  ↳ adding new constraints requires new implementations

Closed Frequent Itemset: State of art

CP approach

→ CP is a generic and declarative framework :

- **Modelling**
  - In a declarative way -> facilitates the addition of new constraints
- **Solving**
  - Efficient solvers based on propagation

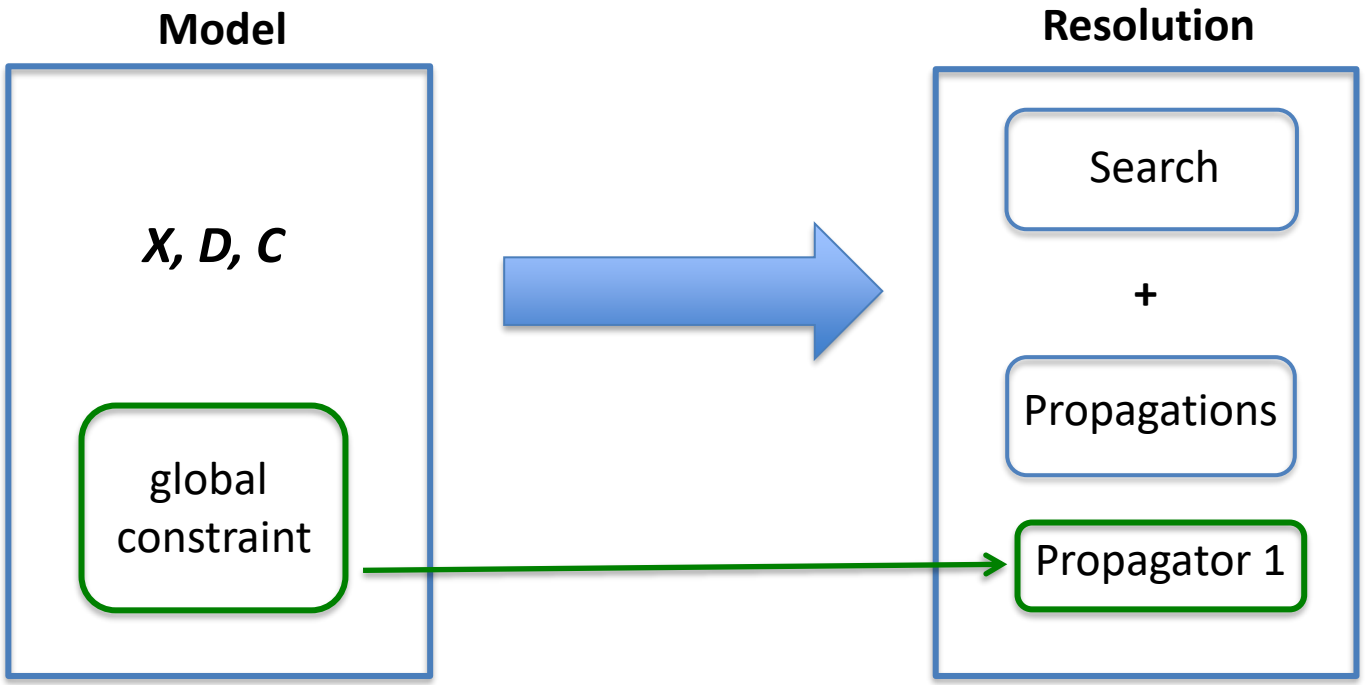
Constraint Programming

CSP :

$C : x_1 + x_2 = x_3$

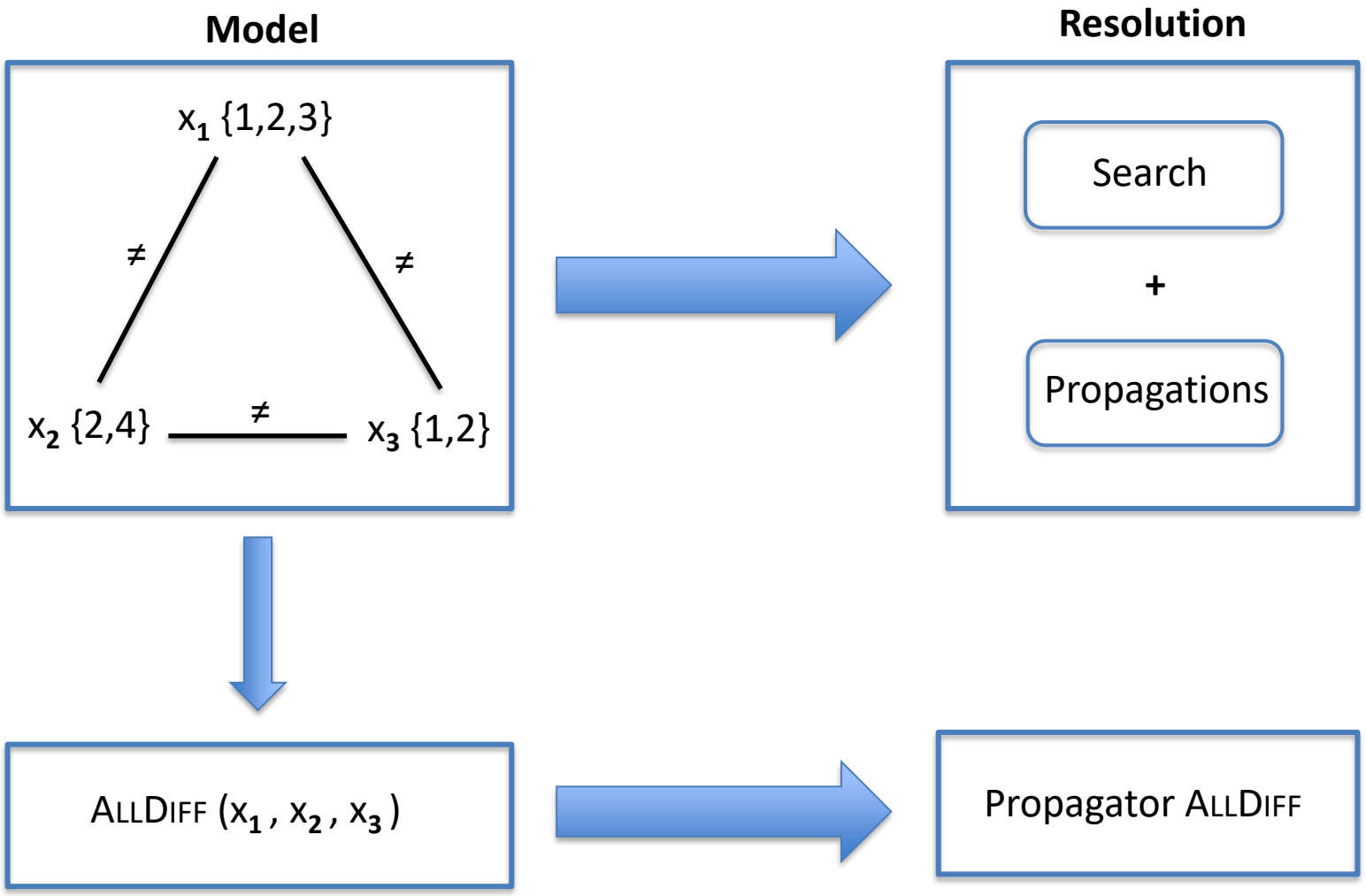
$D(x_1) = \{1,2\}$ $D(x_2) = \{0,2\}$ $D(x_3) = \{3,4\}$

aim: Find values for each x_i



Each global constraint -> dedicated propagator

Global constraint : example



RQ 1 : we use the same schema for itemset mining

Closed Frequent Itemsets : State of art

CP approach

Reified model [De Raedt et al, 08] :

Drawbacks

- Use an additional dimension of transaction variables
- The huge number of constraints: ($2n + m$) reified constraints

Outline

- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* itemset mining for FL
- *F-CPMiner Tool :* implementation & results
- Conclusions & perspectives

Contribution 1 : ClosedPattern

Proposition: a **global constraint** that encodes efficiently the Closed Frequent itemsets mining problem

- Global constraint named ClosedPattern
- Domain consistency with polynomial algorithm
- No reified constraints/extra variables

CLOSEDPATTERN: Definition

Encoding: $P = \{P_1,\ldots,P_n\} \mid D(P_i) = \{0,1\}$

Definition:

$$\text{CLOSEDPATTERN}_{D,\theta}(P) : \text{freq}(P) \geq \theta \;\&\; P \text{ is a closed}$$

trans	items
t1	B C E G H
t2	A D E
t3	A C D H
t4	A D E F
t5	B E F G

P_A	P_B	P_C	P_D	P_E	P_F	P_G	P_H
1	0	0	1	0	0	0	0

$\text{CLOSEDPATTERN}_{D,2}(AD)$ —
- freq(AD) > 2
- AD is closed

CLOSEDPATTERN: Propagation rules

Three Propagation rules

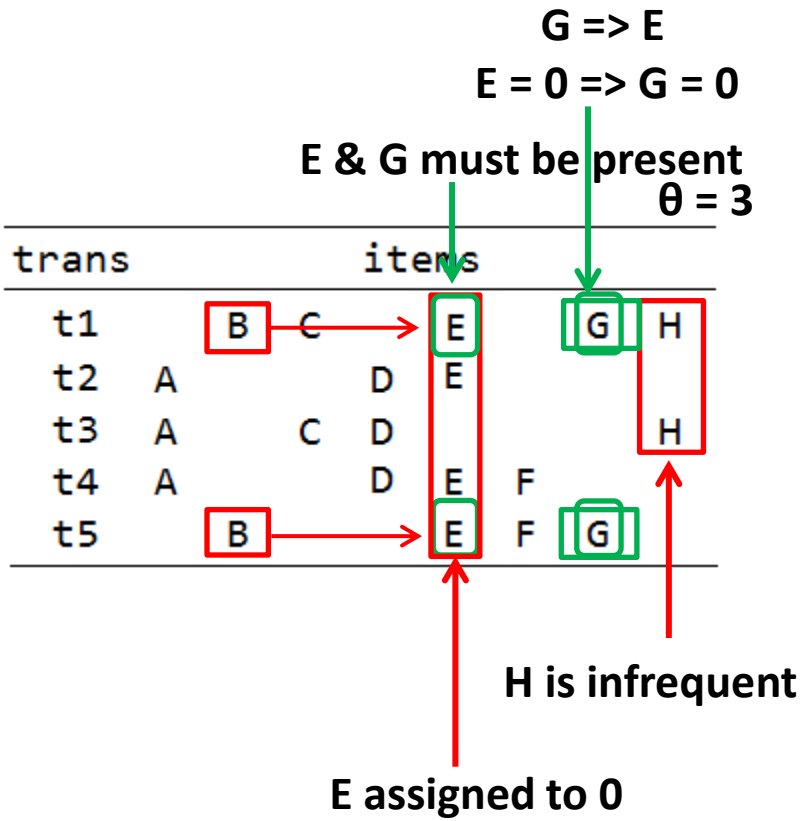
$0 \notin D(P_j)$:

Rule 1: full extension items

$1 \notin D(P_j)$:

Rule 2: infrequent items

Rule 3: missing items



P_A	P_B	P_C	P_D	P_E	P_F	P_G	P_H
0/1	0/1	0/1	0/1	0	0/1	0	0/1

ClosedPattern Propagator

Algorithm: *n*: items, *m*: transactions

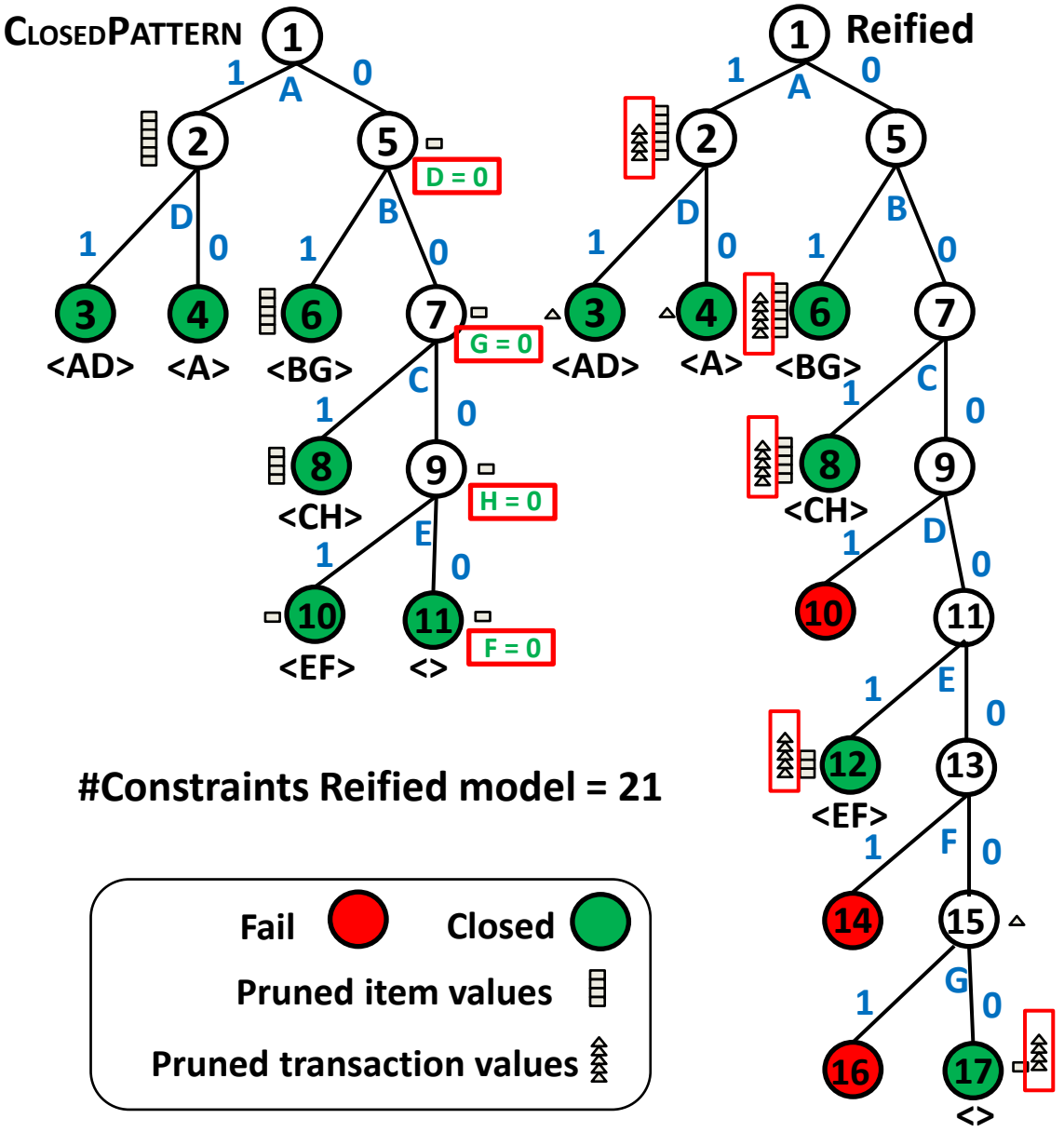
for each free variable

- **rule 1:** maintained (O(*n* x *m*))
- **rule 2:** maintained (O(*n* x *m*))
- for each absent item
  - **rule 3:** maintained (O(*n* x (*n* x *m*)))

Time: *O*(*n* x (*n* x *m*)): Cubic

Space: O(*n* x *m*): Quadratic

ClosedPattern vs Reified



$\theta = 2$

Lex : variables

max_val : values

At each node:

$0 \notin D(P_j)$: rule1

$1 \notin D(P_j)$: rule2

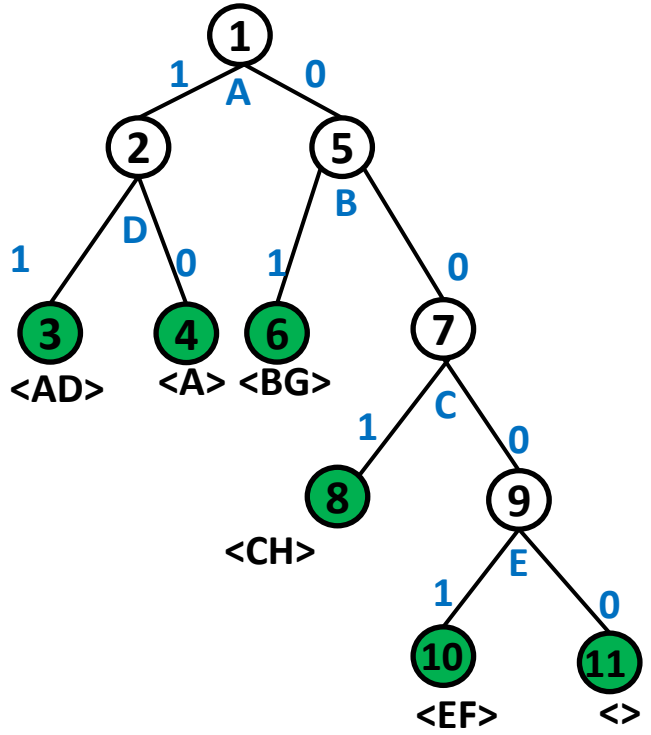
$1 \notin D(P_j)$: rule3

#Constraints Reified model = 21

CLOSEDPATTERN : Complexity

- **Tree search size :**

  2 x|*C*|-1: a closed pattern at each leaf and full binary tree



- **Backtrack-free :**

  $O(|C| \times n^2 \times m)$

ClosedPattern : Experiments

Base	$\|\mathcal{T}\|$	$\|\mathcal{I}\|$	ρ
Chess	3 196	75	49%
Splice1	3 190	287	21%
Mushroom	8 124	119	19%
Connect	67 557	129	33%
BMS-Web-View1	59 602	497	0.5%
T10I4D100K	100 000	1 000	1%
T40I10D100K	100 000	1 000	4%
Pumsb	49 046	7 117	1%
Retail	88 162	16 470	0.06%

- **Comparison with:**
  - The most efficient CP method: CP4IM (reified)
  - The most efficient ad hoc algorithm: LCM-v5.2

- **Solver**: Gecode
- **Data structures :** C++ STL
- **Timeout**= 3600 secs, Intel Xeon E3-1245 V2 @ 3.40 Ghz with 32 Gb

CLOSEDPATTERN : Experiments

CLOSEDPATTERN vs CP4IM :

Dataset	θ	#Patterns	#Nodes		#Failures	
	%		CLOSED PATTERN	CP4IM	CLOSED PATTERN	CP4IM
mushroom	1	51 672	**103 345**	168 999	**0**	32 828
	0.5	76 199	**152 397**	259 427	**0**	53 516
	0.1	164 118	**328 235**	529 289	**0**	100 528
	0.05	203 882	**407 765**	622 145	**0**	107 191
chess	60	98 393	**196 785**	196 787	**0**	1
	50	369 451	**738 901**	738 907	**0**	3
	40	1 366 834	**2 733 667**	2 733 735	**0**	34
retail	5	17	**33**	Memory	**0**	Memory
	1	160	**319**	Memory	**0**	Memory
	0.5	581	**1 161**	Memory	**0**	Memory
	0.1	7 696	**15 391**	Memory	**0**	Memory

density DC Backtrack-free

ClosedPattern: Experiments

ClosedPattern *vs* CP4IM *vs* LCM :

Dataset	θ	*#Patterns*	*#Times*		
	%		ClosedPattern	CP4IM	LCM
mushroom	1	51 672	8.92	21.32	**0.06**
	0.5	76 199	10.32	26.69	**0.09**
	0.1	164 118	12.90	36.99	**0.14**
	0.05	203 882	13.65	39.08	**0.12**
chess	60	98 393	10.88	1.73	**0.04**
	50	369 451	38.59	6.03	**0.14**
	40	1 366 834	133.52	20.95	**0.28**
retail	5	17	0.01	OOM	**0.02**
	1	160	0.10	OOM	**0.02**
	0.5	581	0.49	OOM	**0.03**
	0.1	7 696	38.10	OOM	**0.09**

LCM still the best for mining **Closed frequent** itemset

CLOSEDPATTERN: Experiments

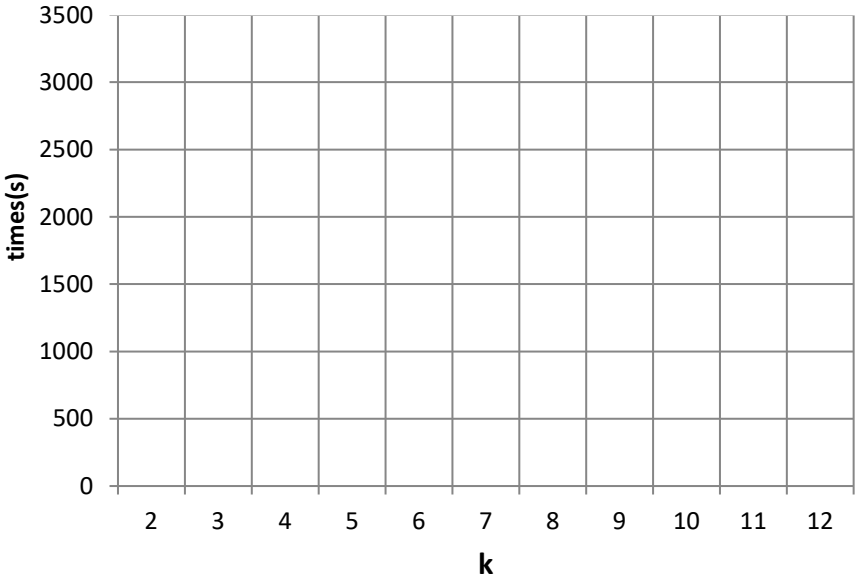
k patterns set

The aim : find *k* closed itemsets:

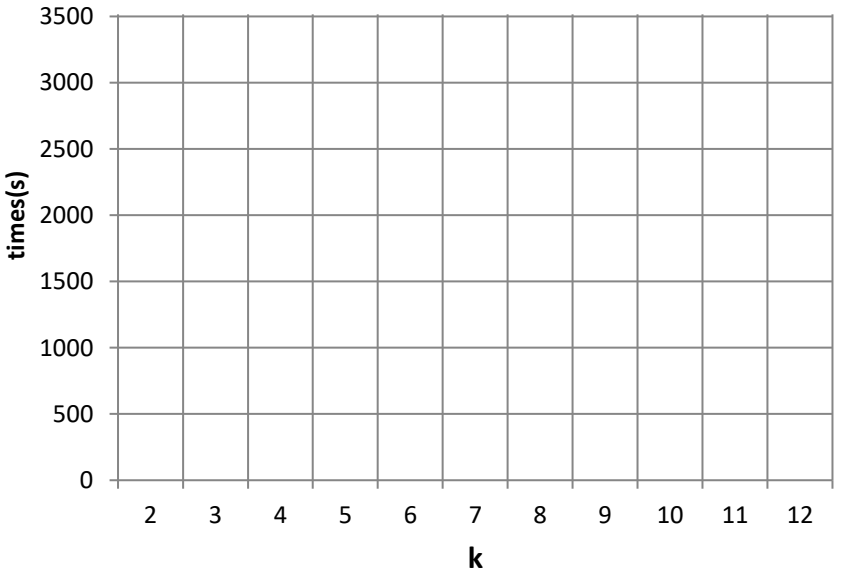
CLOSEDPATTERN

Distinct itemsets

lb < *size* < *ub*



chess (θ = 80%, *lb* = 2, *ub* = 10)



connect (θ = 90%, *lb* = 2, *ub* = 10)

CLOSEDPATTERN: Experiments

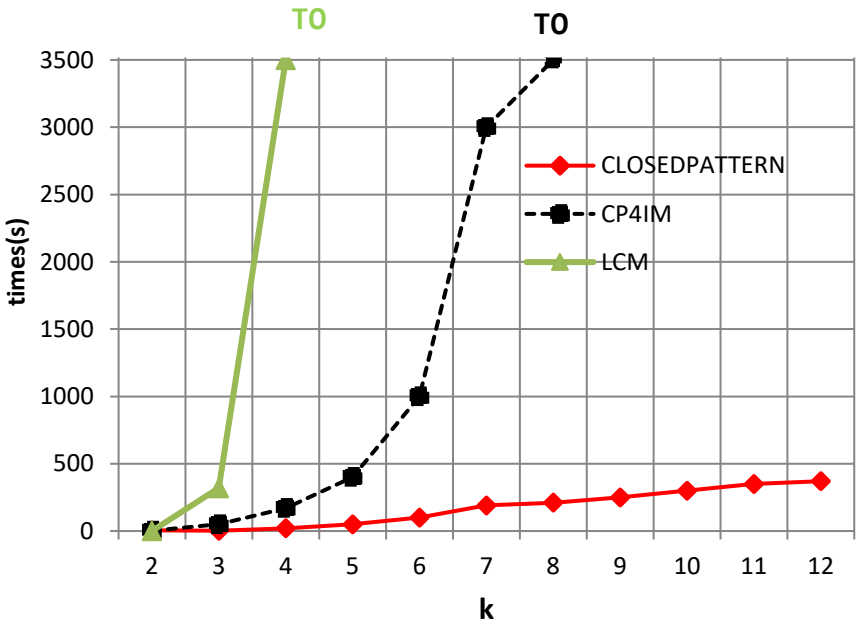
k patterns set

The aim : find *k* closed itemsets:

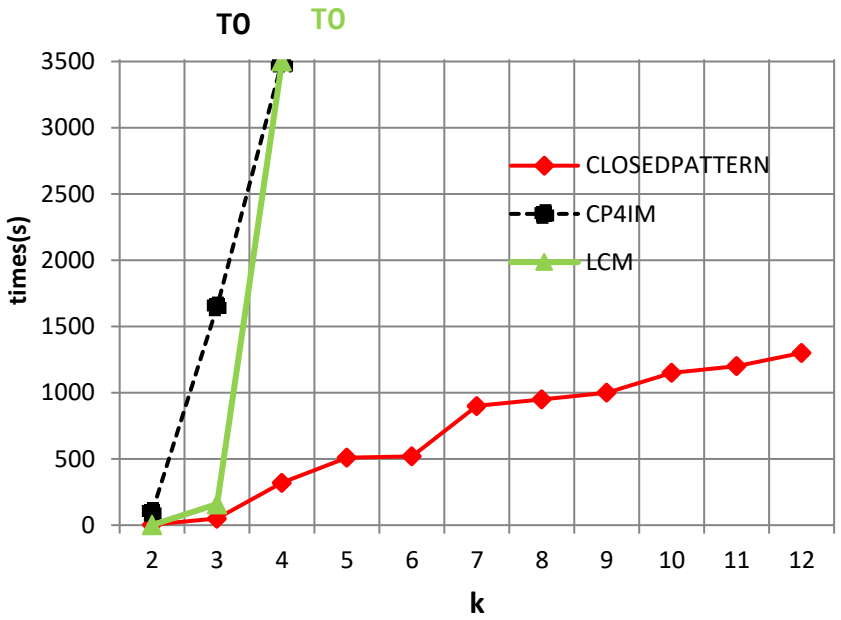
CLOSEDPATTERN

Distinct itemsets

lb < *size* < *ub*



chess (θ = 80%, *lb* = 2, *ub* = 10)



connect (θ = 90%, *lb* = 2, *ub* = 10)

Conclusions

- A global constraint for Closed Frequent Itemset ensuring DC

- No need for reified constraints/extra variables

- Filtering algorithm cubic in time, quadratic in space

Outline

- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* itemset mining for FL
- Contribution 3 : *F-CPMiner* Tool
- Conclusions & perspectives

Software testing : defintion

- Software testing: Process of evaluating a system (program) with the intent to check if it respects its specifications (Oracle)

Three main purposes:

Software testing : Fault localization

The need: identify a subset of statements that are susceptible to explain the origin of the errors of a program

Precision ↔ efficiency

Some approaches, based on:

Execution traces *[Renieres et al., 93]*

Suspicion measures *[Jones et al.,09]*

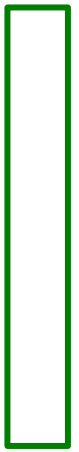
Data mining *[Cellier et al., 05]*

Fault Localization : Example

- **Elements of a program under test:**

Program : Character counter

```
function count (char *s) {
     int let, dig, other, i = 0;
     char c;
e1:  while (c = s[i++]) {
e2:    if('A'<=c && 'Z'>=c)
e3:      let += 2; //- fault -
e4:    else if ( 'a'<=c && 'z'>=c )
e5:      let += 1;
e6:    else if ( '0'<=c && '9'>=c )
e7:      dig += 1;
e8:    else if (isprint (c))
e9:      other += 1;
e10:  printf("%d %d %d\n", let, dig, other);}
```



Test case: $tc_i = (D_i , O_i)$: *Passing/Failing*

Test suite: $T = \{tc_1 \ldots tc_8\}$

Test case coverage: statements executed at least once

Fault localization : *Suspicious statement*

- Statement that appears more frequently in failing test cases and less in passing ones

- Each *method/measure* tries to capture the notion of suspiciousness

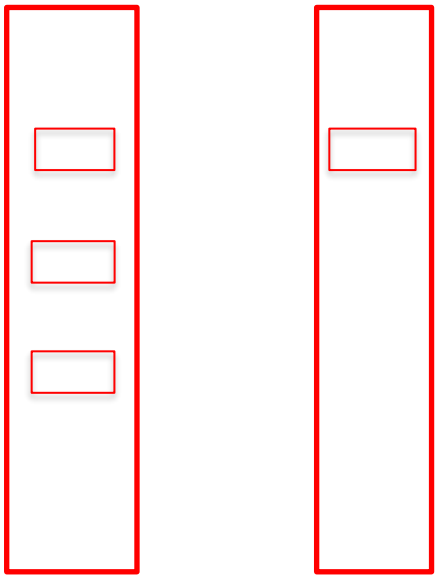
Aim: Total ordering of statements from the most suspicious to guiltless one

Suspicion measures : *Example*

TARANTULA [Jones et al, 05] & **OCHIAI** [Abreu et al, 07]

failing **passing**

	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7	tc_8
e_1	1	1	1	1	1	1	1	1
e_2	1	1	1	1	1	1	0	1
e_3	1	1	1	1	1	1	0	0
e_4	1	1	1	1	1	0	0	1
e_5	1	1	0	0	1	0	0	0
e_6	1	1	1	1	0	0	0	1
e_7	0	1	0	1	0	0	0	0
e_8	1	0	1	0	0	0	0	1
e_9	1	0	1	0	0	0	0	1
e_{10}	1	1	1	1	1	1	1	1



Each measure gives a specific ranking

Fault localization : *Suspiciousness Measures*

Advantage:

- Quick evaluation of each statement

Drawbacks:

- Evaluating statements *individually* and *independently* of each other => Ignore the *dependencies*

Can we exploit the dependencies between executions?

Fault localization using declarative itemset mining

Outline

- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* itemset mining for FL
- Contribution 3 : *F-CPMiner* Tool
- Conclusions & perspectives

Contribution 2

There exist dependencies between statements/test cases

how to proceed ?

Our idea:

Extract a set of suspicious itemsets (set of statements)

CP approach:

A generic tool catching various needs of fault localization

Our proposed method: *dataset*

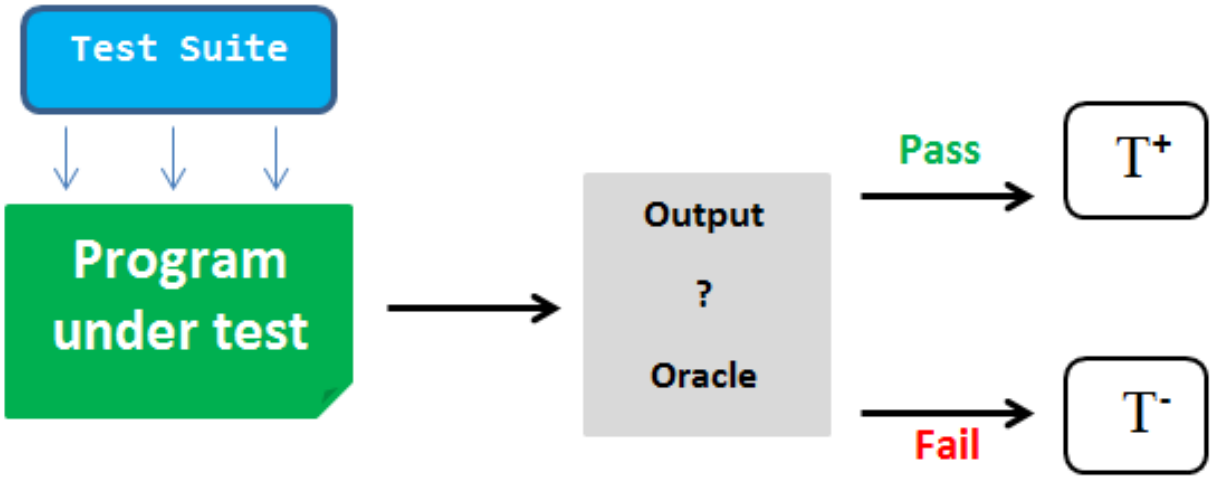
Test suite coverage = transactional database

Statements	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7	tc_8
e_1 : while (c = s[i++]) {	1	1	1	1	1	1	1	1
e_2 : if('A'<=c && 'Z'>=c)	1	1	1	1	1	1	0	1
e_3 : let += 2; // -fault -	1	1	1	1	1	1	0	0
e_4 : else if ('a'<=c && 'z'>=c)	1	1	1	1	1	0	0	1
e_5 : let += 1;	1	1	0	0	1	0	0	0
e_6 : else if ('0'<=c && '9'>=c)	1	1	1	1	0	0	0	1
e_7 : dig += 1;	0	1	0	1	0	0	0	0
e_8 : else if (isprint (c))	1	0	1	0	0	0	0	1
e_9 : other += 1;	1	0	1	0	0	0	0	1
e_{10} : printf("%d %d %d\n", let, dig, other);}	1	1	1	1	1	1	1	1

- each statement e_i corresponds to an item
- each test case coverage tc_i forms a transaction

Our proposed method: *dataset*

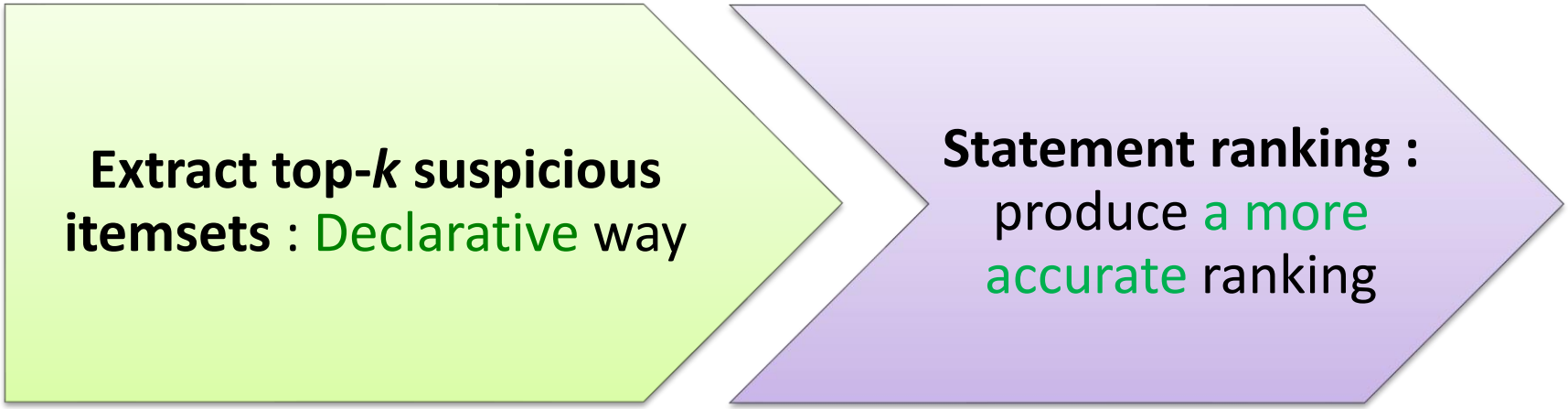
The transactional database is partitioned in 2 classes:



The aim: Extract relevant itemsets (contrast)

Our proposed method

Steps and aims:



Our approach -> suspicious itemset S

Frequency:

The itemset *S* must appear at least once in T^- : $freq^-(S) \geq 1$

Closedness:

The largest itemset for a given degree of suspiciousness

$$\text{CLOSEDPATTERN}_{T,\theta}(S), \text{ such that } T = T^+ \cup T^-$$

Fault Localization : *top-k suspicious itemsets*

Dominance relation :

$S \succ_R S'$ iff:

$$[freq^-(S) > freq^-(S')] \vee$$
$$[(freq^-(S) = freq^-(S')) \wedge (freq^+(S) < freq^+(S'))]$$

produce

top-k suspicious itemsets

Fault Localization : *top-k suspicious itemsets*

Algorithm top-*k* most suspicious itemsets :

- Input : T^-, T^+, k
- Extract the k first itemset $S = (S_1,...,S_k)$

  > Ordering S in decreasing order according to $\succ_R$

- Exploit $\succ_R$ → Produce a continous refinement

  > The constraint ($S' \succ_R S_k$) is posted
  >
  > Update S

- Output : top-*k* most suspicious itemsets

top-k suspicious itemsets : *example*

top-k suspicious itemsets	freq+	freq-
S1:<e1,e2,e3,e10>	0	6
S2:<e1,e2,e10>	1	6
S3:<e1,e10>	2	6
S4:<e1,e2,e3,e4,e10>	0	5
S5:<e1,e2,e4,e10>	1	5
S6:<e1,e2,e3,e4,e6,e10>	0	4
S7:<e1,e2,e4,e6,e10>	1	4
S8:<e1,e2,e3,e4,e5,e10>	0	3
S9:<e1,e2,e3,e4,e6,e7,e10>	0	2
S10:<e1,e2,e3,e4,e6,e8,e9,e10>	0	2

- Each S_i : subset of statements that can locate de faulty statement

- 1st localization: but itemsets can be quite large -> refine the result

Step2 : statements ranking

- **Observations and rules:**

top-k suspicious itemsets	freq+	freq-
S1:<e1,e2,e3,e10>	0	6
S2:<e1,e2,e10>	1	6
S3:<e1,e10>	2	6
S4:<e1,e2,e3,e4,e10>	0	5
S5:<e1,e2,e4,e10>	1	5
S6:<e1,e2,e3,e4,e6,e10>	0	4
S7:<e1,e2,e4,e6,e10>	1	4
S8:<e1,e2,e3,e4,e5,e10>	0	3
S9:<e1,e2,e3,e4,e6,e7,e10>	0	2
S10:<e1,e2,e3,e4,e6,e8,e9,e10>	0	2

S_1 : the most suspicious -> most susceptible to contain the fault

Step2 : statements ranking

- **Observations and rules:**

top-k suspicious itemsets	freq+	freq-
S1:<e1,e2,e3,e10>	0	6
S2:<e1,e2,e10>	1	6
S3:<e1,e10>	2	6
S4:<e1,e2,e3,e4,e10>	0	5
S5:<e1,e2,e4,e10>	1	5
S6:<e1,e2,e3,e4,e6,e10>	0	4
S7:<e1,e2,e4,e6,e10>	1	4
S8:<e1,e2,e3,e4,e5,e10>	0	3
S9:<e1,e2,e3,e4,e6,e7,e10>	0	2
S10:<e1,e2,e3,e4,e6,e8,e9,e10>	0	2

From an itemset S_i to S_{i+1} some statements appear/disappear

3 categories of statements in top-*k* suspicious itemsets

step2: statements ranking

- **Observations and rules:**

top-k suspicious itemsets	freq+	freq-
S1:<e1,e2,e3,e10>	0	6
S2:<e1,e2,e10>	1	6
S3:<e1,e10>	2	6
S4:<e1,e2,e3,e4,e10>	0	5
S5:<e1,e2,e4,e10>	1	5
S6:<e1,e2,e3,e4,e6,e10>	0	4
S7:<e1,e2,e4,e6,e10>	1	4
S8:<e1,e2,e3,e4,e5,e10>	0	3
S9:<e1,e2,e3,e4,e6,e7,e10>	0	2
S10:<e1,e2,e3,e4,e6,e8,e9,e10>	0	2

Statements $\in S_1$ and disappear in S_i (i=*2*..*k*)

Ω_1: most suspicious

step2 : statements ranking

- **Observations and rules:**

top-k suspicious itemsets	freq+	freq-
S1:<e1,e2,e3,e10>	0	6
S2:<e1,e2,e10>	1	6
S3:<e1,e10>	2	6
S4:<e1,e2,e3,e4,e10>	0	5
S5:<e1,e2,e4,e10>	1	5
S6:<e1,e2,e3,e4,e6,e10>	0	4
S7:<e1,e2,e4,e6,e10>	1	4
S8:<e1,e2,e3,e4,e5,e10>	0	3
S9:<e1,e2,e3,e4,e6,e7,e10>	0	2
S10:<e1,e2,e3,e4,e6,e8,e9,e10>	0	2

Statements that belong to all S_i (i=*1..k*)

Ω_2: neutral

step2 : statements ranking

- **Observations and rules:**

top-k suspicious itemsets	freq+	freq-
S1:<e1,e2,e3,e10>	0	6
S2:<e1,e2,e10>	1	6
S3:<e1,e10>	2	6
S4:<e1,e2,e3,e4,e10>	0	5
S5:<e1,e2,e4,e10>	1	5
S6:<e1,e2,e3,e4,e6,e10>	0	4
S7:<e1,e2,e4,e6,e10>	1	4
S8:<e1,e2,e3,e4,e5,e10>	0	3
S9:<e1,e2,e3,e4,e6,e7,e10>	0	2
S10:<e1,e2,e3,e4,e6,e8,e9,e10>	0	2

Statements $\notin S_1$ and appear gradually in S_i (i=*2..k*)

Ω_3: Guiltless

Statements ranking

Ranking = < Ω_1 , Ω_2 , Ω_3 >

Statements	Rank	List
e3	1	Ω_1
e2	2	Ω_1
e1,e10	4	Ω_2
e4	5	Ω_3
e6	6	Ω_3
e5	7	Ω_3
e7	8	Ω_3
e8,e9	10	Ω_3

Contribution 2 : Conclusions

A new approach based itemset mining for FL

Exploit dependencies between statements/test cases

Approach on two steps :

- top-*k* suspicious patterns extraction
- ad-hoc algorithm for an accurate ranking

Outline

- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* IM for FL
- Contribution 3 : *F-CPMiner* Tool
- Conclusions & perspectives

Experiments : *Implementations*

F-CPMINER Tool :

Extract top-*k* suspicious itemsets : Declarative way

+

Ad-hoc algorithm for statement ranking

Version 1 : using the reified model

Version 2: using our global constraint CLOSEDPATTERN

Resolution : GECODE

Implementation : C++

Experiments : *Benchmark programs*

Siemens suite : 111 programs

Program	#Faulty versions	L.O.C	#Test cases
Replace	29	514	5542
Print Tk2	9	358	4056
Print Tk	4	348	4071
Schedule	4	294	2650
Schedule2	8	265	2680
Tot Info	19	272	1015
Tcas	37	135	1542

Proc Intel Xeon CPU E3-1245 V2 with 32Go RAM

Efficiency measure : *ExamScore* (% of code to examine)
P-Exam, O-Exam

Experiments : *Performances comparison*

Comparing both models based on:

Global constraint : **F-CPMiner-V1**

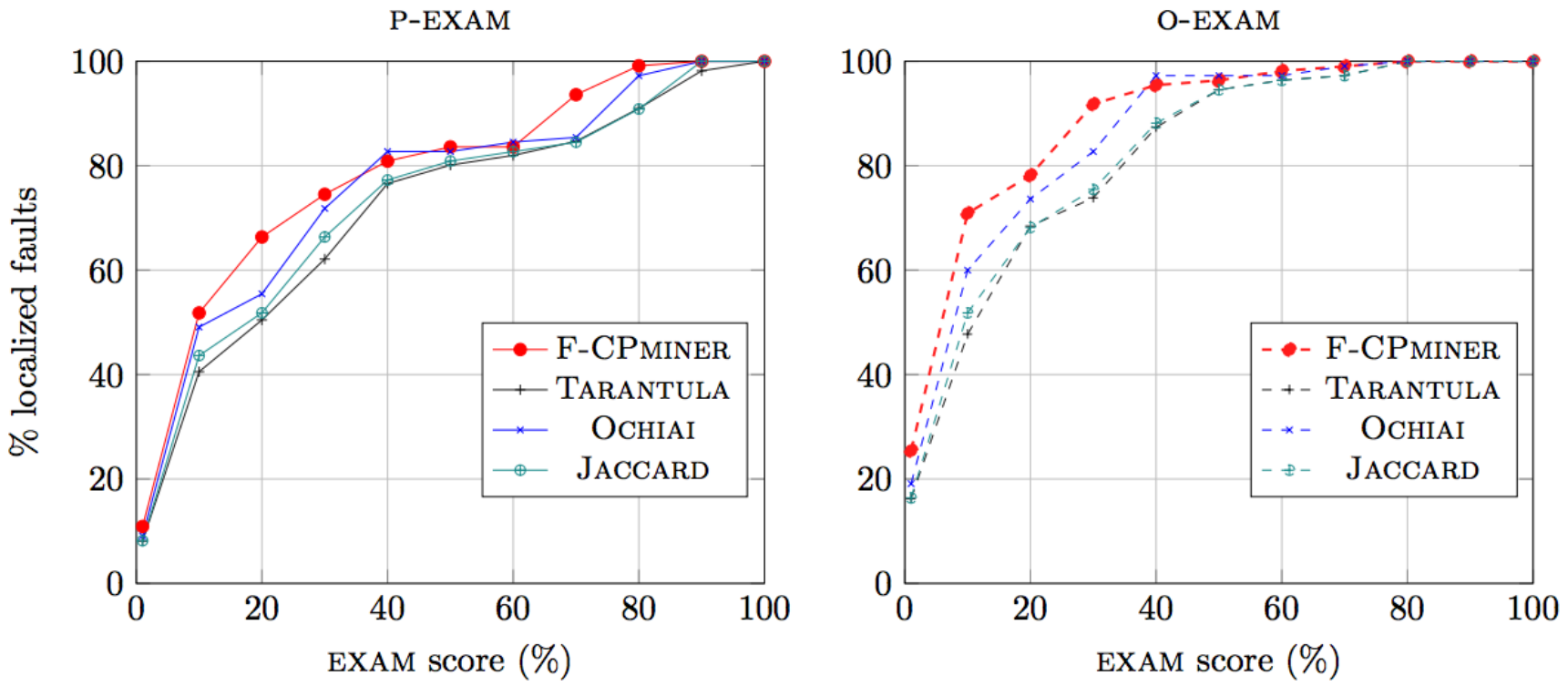
Reified Model : **F-CPMiner-V2**

		CPU time			#Propagations		#Nodes	
Prog	k	F-CPMINER-V1	F-CPMINER-V2	Speed up	F-CPMINER-V1	F-CPMINER-V2	F-CPMINER-V1	F-CPMINER-V2
(1)	245	**20.03**± 20.46	147.69 ± 86.02	**7.37**	**8116650**	216625216	**37530**	2450119
(2)	200	**4.62** ± 0.47	146.25 ± 65.89	**31.65**	**2590350**	180761014	**5468**	3588108
(3)	195	**4.62** ± 0.39	61.49 ± 34.35	**13.28**	**1914429**	70013851	**3331**	1235562
(4)	152	**0.85** ± 0.26	27.41 ± 15.28	**32.24**	**489246**	15821395	**2350**	210884
(5)	128	**1.46** ± 1.78	12.66 ± 5.37	**8.67**	**807141**	7602101	**4870**	113438
(6)	123	**0.20** ± 0.07	2.53 ± 1.01	**12.65**	**229505**	1698834	**1023**	10445
(7)	65	**0.04** ± 0.01	0.16 ± 0.02	**4**	**99241**	245331	**60**	133

(1) Replace, (2) Print tokens2, (3) Print Tokens, (4) Schedule, (5) Schedule2, (6) Tot info, (7) Tcas

Timeout = 180 seconds

Experiments : *Effectiveness comparison*



- **F-CPMiner** is highly competitive.

- On **99%** of processed programs, the fault was on the pattern S_1

Experiments : *Effectiveness comparison*

Total examined statements:

Siemens suite contains : 16k statements

	F-CPMiner	Taruntula	Jaccard	Ochiai
P-EXAM	**14.58 %**	21.47 %	17.69 %	20.85 %
O-EXAM	**8.46 %**	15.36 %	11.57 %	14.74 %

Outline

- Context & Motivations
- Itemset mining
- *Contribution 1 :* A global constraint for CFPM
- Software testing & Fault Localization
- *Contribution 2 :* IM for FL
- Contribution 3 : *F-CPMiner* Tool
- Conclusions & perspectives

Conclusions

Declarative Itemset mining :

- A global constraint for Closed Frequent Itemset ensuring DC
- No need for reified constraints/extra variables
- Filtering algorithm cubic in time, quadratic in space.

Fault localization :

- A new approach based on top-k suspicious patterns
- The model uses the ClosedPattern global constraint
- A new tool named *F-CPMiner*

Perspectives

Declarative Itemset mining :

- Improve the complexity/efficiency of the global constraint
- Extend the global constraint for new type of itemsets
- New platform for declarative data mining (in progress)

Fault localization :

- Multiple faults localization (submitted CP'17)
- Explore more observations on behavior of faulty program
- Using sequence mining

Publications

A global constraint for frequent closed itemsets mining

- *M. Maamar, C. Bessiere, P. Boizumault, N. Lazaar, Y. Lebbah, V. Lemière and S. Loudni*. CLOSEDPATTERN : Une contrainte globale pour l'extraction de motifs fréquents fermés. *JFPC'17*

- *N. Lazaar, Y. Lebbah, S. Loudni, M. Maamar, V. Lemiere, C. Bessiere, and P. Boizumault*. A global constraint for closed itemset mining. *CP'16*.

Itemset mining for fault localization problem

- *M. Maamar, N. Lazaar, Y. Lebbah and S. Loudni*. Fault localization using itemset mining under constraints. *ASE Journal'16*

- *M. Maamar, N. Lazaar, Y. Lebbah and S. Loudni*. *F-CPMiner* : Une approche pour la localisation de fautes basée sur l'extraction de motifs ensemblistes sous contraintes. *JFPC'16*

- *M. Maamar, N. Aribi, N. Lazaar, Y. Lebbah and S. Loudni*. F-*CPMiner** : a new approach for fault localization using itemset mining and constraint programming. *CP meets Verif CP Workshop'16*

Merci pour vos ±45 minutes d'attention…

Questions….