Identifying and Generating Missing Tests using Machine Learning on Execution Traces

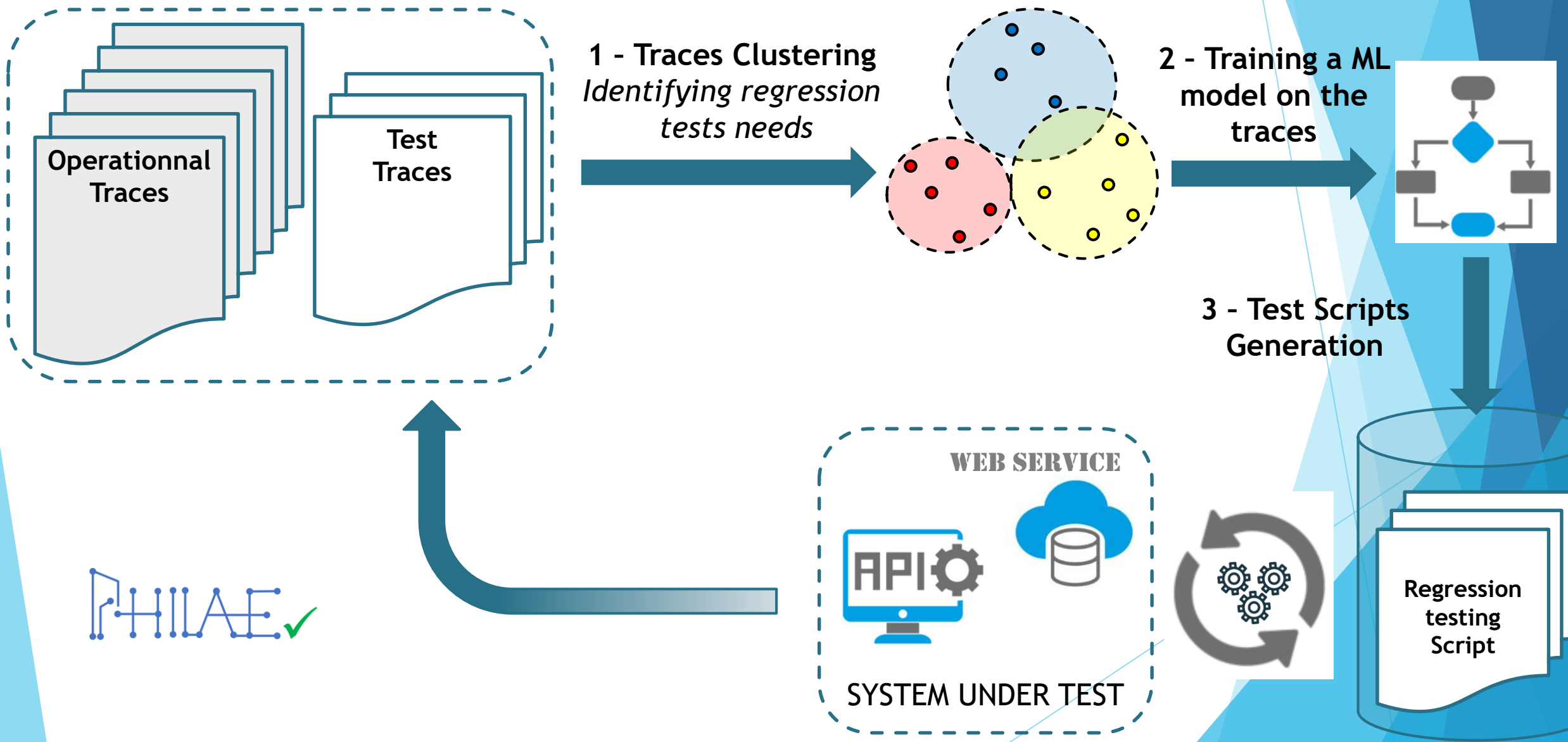
Mark Utting – Bruno Legeard – Frédéric Dadeau – Frédéric Tamagnan
Fabrice Bouquet

This work was supported in part by the French National Research Agency: PHILAE project (N° ANR-18-CE25-0013)

Motivations

- Efforts to automate tests are increasingly creating a project bottleneck (effort, time, ressources) and a strong reliance
- The growing interest for AI in the testing field
- The growing capacity of logs storage

PHILAE project outline :



Main contributions of this paper :

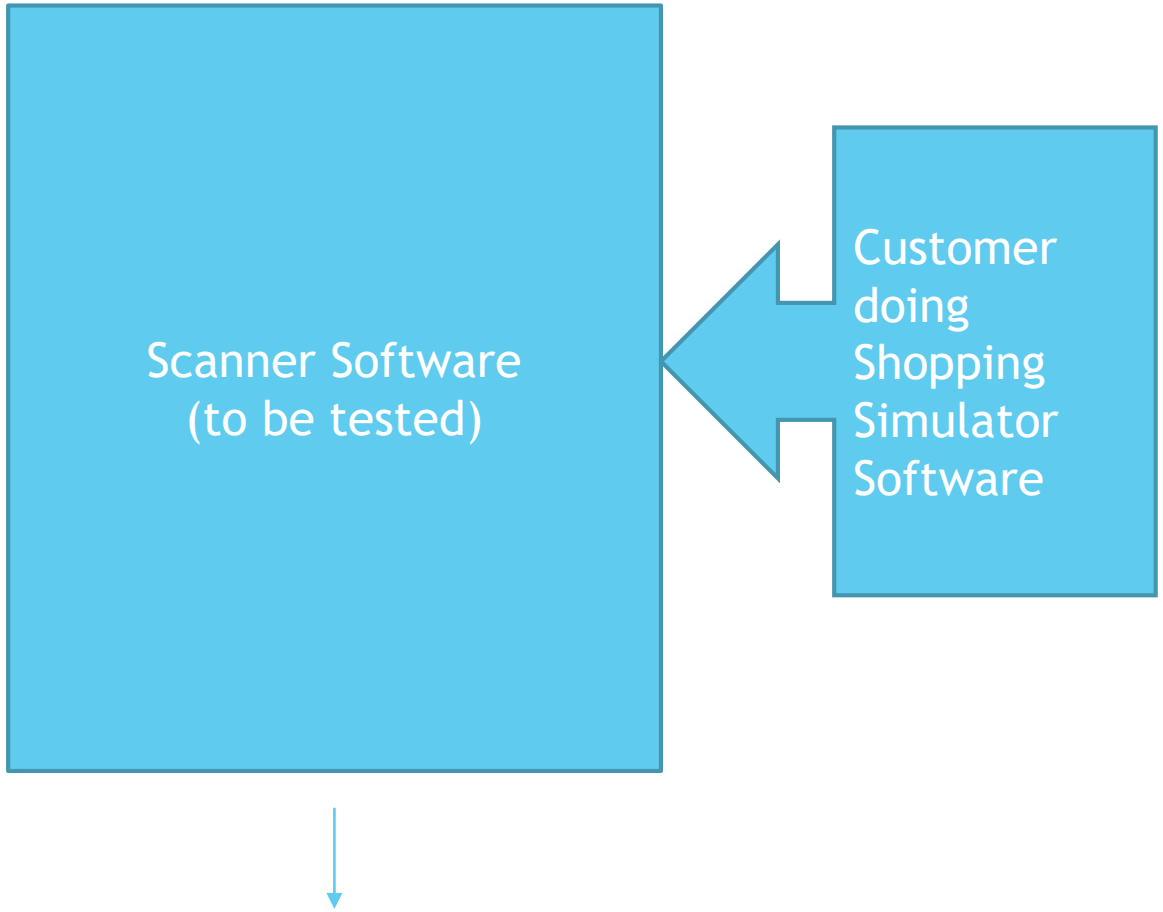
1. Identifying regression test needs by comparing test execution traces and operational execution traces using clustering and visualisation techniques
2. Automating test generation using a predictive machine learning model of user traces to propose new test cases covering the identified regression test needs
3. An open-source toolbox supporting these services
4. Experimental evaluation on two industry web services

Running Example : Supermarket Scanner

Running Example : Supermarket Scanner

Definition : device that is able to read products barcodes and store them into a shopping list

Running Example : Supermarket Scanner



Running Example : Supermarket Scanner

List of possible actions for a customer doing shopping :

1. *Unlock a scanner for shopping*
2. *Scan an article (as the customer put it into their physical basket)*
3. *Delete an article*
4. *Transmission to the checkout*
5. *Abandon the scanner*

Running Example : Supermarket Scanner

Non-nomical cases :

1. *The customer could scan an unknown bardcode. The article is added to the shopping list, but the cashier will have to add it manually later*
2. *The customer could be asked to a control check and have to re-scan the articles after the transmission*

Running Example : Supermarket Scanner

List of possible actions for the cashier during the checkout:

1. *Open a session*
2. *Add an article*
3. *Remove an article*
4. *Close the session*
5. *Make the customer pay*

Running Example : Supermarket Scanner

```
# timestamp, sessID, object, action, inputs, output
1570573649196, 41, scan3, abandon, [], 0
1570573649191, 42, scan1, transmit, [checkout0], 0
1570573649197, 42, scan1, abandon, [], 0
1570573649355, 43, scan1, unlock, [], 0
1570573649358, 42, checkout0, openSession, [], 0
1570573649996, 43, scan1, scan, [5410188006711], 0
1570573650366, 43, scan1, scan, [5410188006711], 0
1570573650366, 42, checkout0, add, [3570590109324], 0
1570573650389, 44, scan2, scan, [3046920010856], 0
1570573651369, 42, checkout0, closeSession, [], 0
1570573652376, 42, checkout0, pay, [68.27], 0
1570573655132, 40, scan0, scan, [7640164630021], -2
1570573656245, 44, scan2, scan, [3270190022534], 0
1570573656633, 43, scan1, scan, [3474377910724], 0
...
```

Fig. 1. Raw data of the system log file.

- Our logs are composed by 65000+ steps (actions) from 4518 traces.

README.md

Agilkia: A Python Toolkit to Support AI-for-Testing

This toolkit is intended to make it easier to build testing tools that learn from traces of customer behaviors, analyze those traces for common patterns and unusual behaviors (e.g. using clustering techniques), learn machine learning (ML) models of typical behaviors, and use those models to generate smart tests that imitate customer behaviors.

Agilkia is intended to provide a storage and interchange format that makes it easy to built 'smart' tools on top of this toolkit, often with just a few lines of code. The main focus of this toolkit is saving and loading traces in a standard *.JSON format, and transforming those traces to and from lots of other useful formats, including:

Trace preprocessing

https://github.com/utting/agilkia

Traces preprocessing

- Loading csv into a Agilkia « Trace » object that contains a sequence of « Event » objects
- Splitting and grouping the event by users session thanks to the user ID

Traces preprocessing

- Visualization of traces by mapping a letter to each method name (Unlock -> u, Scan -> . , delete -> d, etc)

```
scan1, unlock, [], 0
scan1, scan, [3474377910724], 0
scan1, scan, [3046920010856], 0
scan1, delete, [3046920010856], 0
scan1, scan, [7640164630021], -2
scan1, transmit, [checkout2], 0
scan1, abandon, [], 0
checkout2, openSession, [], 0
checkout2, add, [7640164630021], 0
checkout2, closeSession, [], 0
checkout2, pay, [14.95], 0
```

u..d.tao+cp

(summarized view of each trace)

Traces preprocessing

- Vectorization of the traces thanks to the bag-of-words representation

```
scan1, unlock, [], 0
scan1, scan, [3474377910724], 0
scan1, scan, [3046920010856], 0
scan1, delete, [3046920010856], 0
scan1, scan, [7640164630021], -2
scan1, transmit, [checkout2], 0
scan1, abandon, [], 0
checkout2, openSession, [], 0
checkout2, add, [7640164630021], 0
checkout2, closeSession, [], 0
checkout2, pay, [14.95], 0
```

abandon	1
add	1
closeSession	1
delete	1
openSession	1
pay	1
scan	3
transmit	1
unlock	1

Fig. 2. A customer trace and its Bag of Words vectorization

Trace clustering, visualization and test need identification

Traces clustering

- Clustering of 4818 customer traces with MeanShift Algorithm with bag-of-words vectorization.
- We compare the clustering of operational traces (4518) and test traces (30)

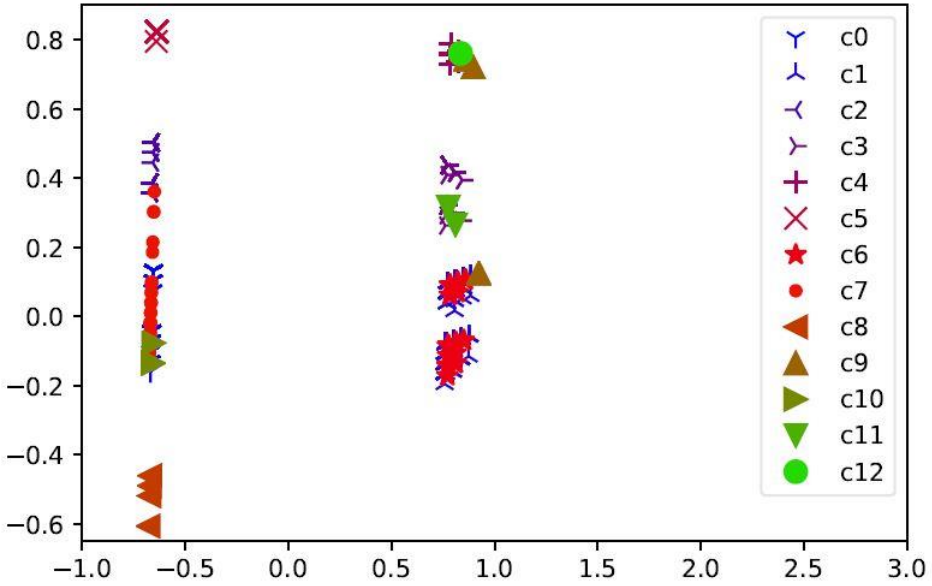


Fig. 4. Visualization of the 4818 customer traces in 13 clusters.

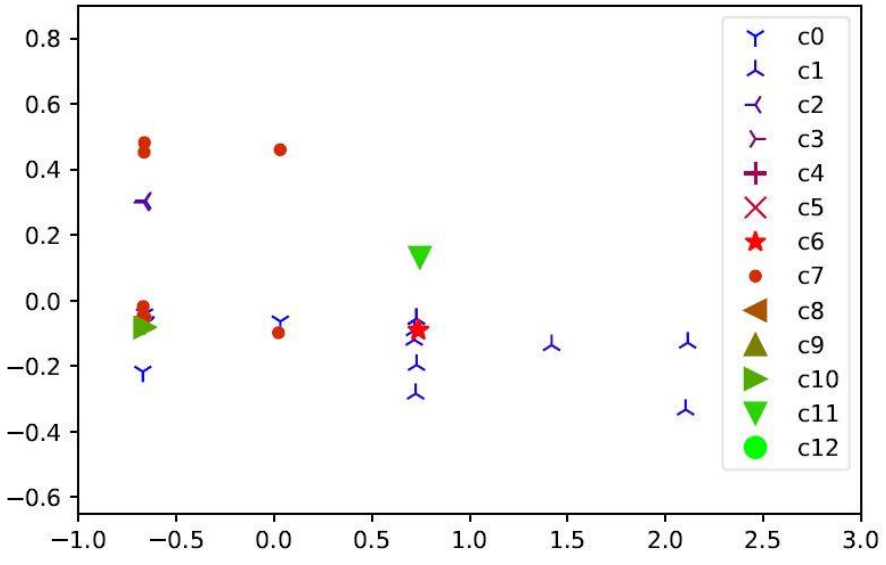


Fig. 5. Visualization of the 30 system test traces, mapped into the same 2D PCA space and clusters as the customer traces.

$$X = 0.69 * \#openSession + 0.19 * \#add + 0.69 * \#closeSession$$
$$Y = 0.73 * \#scan + 0.68 * \#transmission$$

Traces clustering

- The clusters represent the different behaviors that have been implemented in the scanner simulator

Id	Size	Description
0	2314	classical usage of the scanner (no control check, direct payment)
1	1996	shopping with unknown references that are added afterwards by the cashier before payment
2	218	shopping followed by a control check by the cashier then payment
3	129	similar to 2 with the addition of unknown products afterwards
4	45	similar to 3 but with longer sequences (more products to scan and control)
5	40	similar to 2 but with longer sequences (more products to scan and control)
6	35	same as 1 but with removals of products during the shopping
7	26	shopping, followed by a control which fails (detects a product that had not been scanned)
8	4	the sequences that were interrupted at the end of the log file
9	5	shopping, followed by a control, or not, followed by a manual addition of product by the cashier
10	3	short shopping with removal and direct payment without control
11	2	short shopping with removal and some manual additions of products
12	1	full complete sequence (shopping, control, addition of products)

TABLE I

RESULT OF THE CLUSTERING ON THE CUSTOMER TRACES

Traces clustering

- That allows us to identify the testing needs

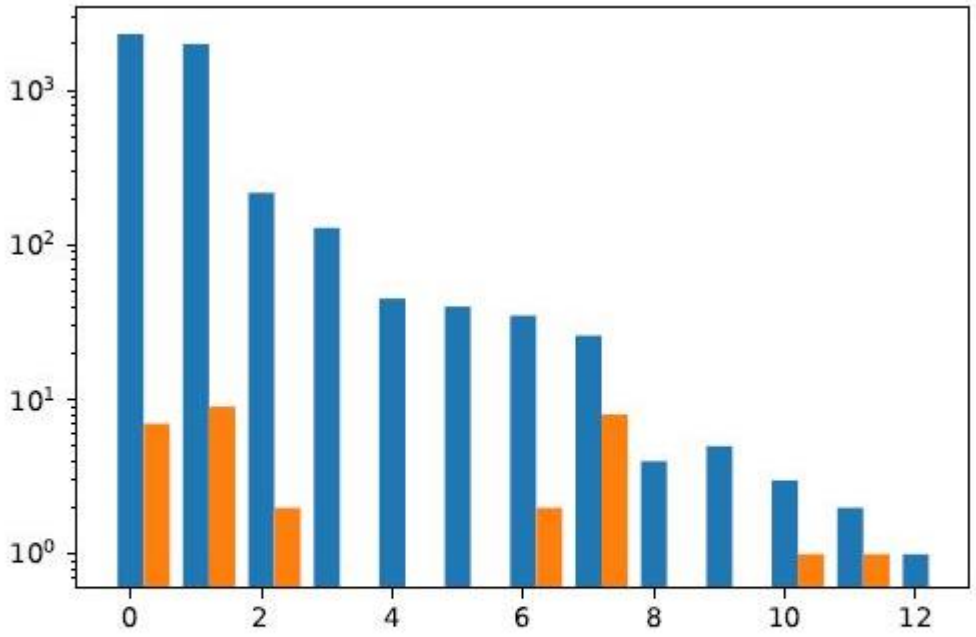


Fig. 3. Number of customer traces in each cluster (left/blue) compared with number of test traces (right/orange) in the same cluster. Y-axis is log scale.

Test generation using a predictive ML model

Learning a model to predict the next action

- Considering this trace : u.....t.....tao+cp

- It gives us 21 couples of (prefix, next action) to train a ML model :
  - (u,.)
  - (u.,.)
  - …
  - (u.....t.....tao+c,p)
  - (u.....t.....tao+cp,<END>)

- We can take all the couples (prefix,next action) from a specific cluster which has no system test to train a model that generate sequences in this manner.

Learning a model to predict the next action

- We trained several classical ML model (Random Forests, Gradient Boosting, etc) over different clusters traces with 10-fold cross validation.

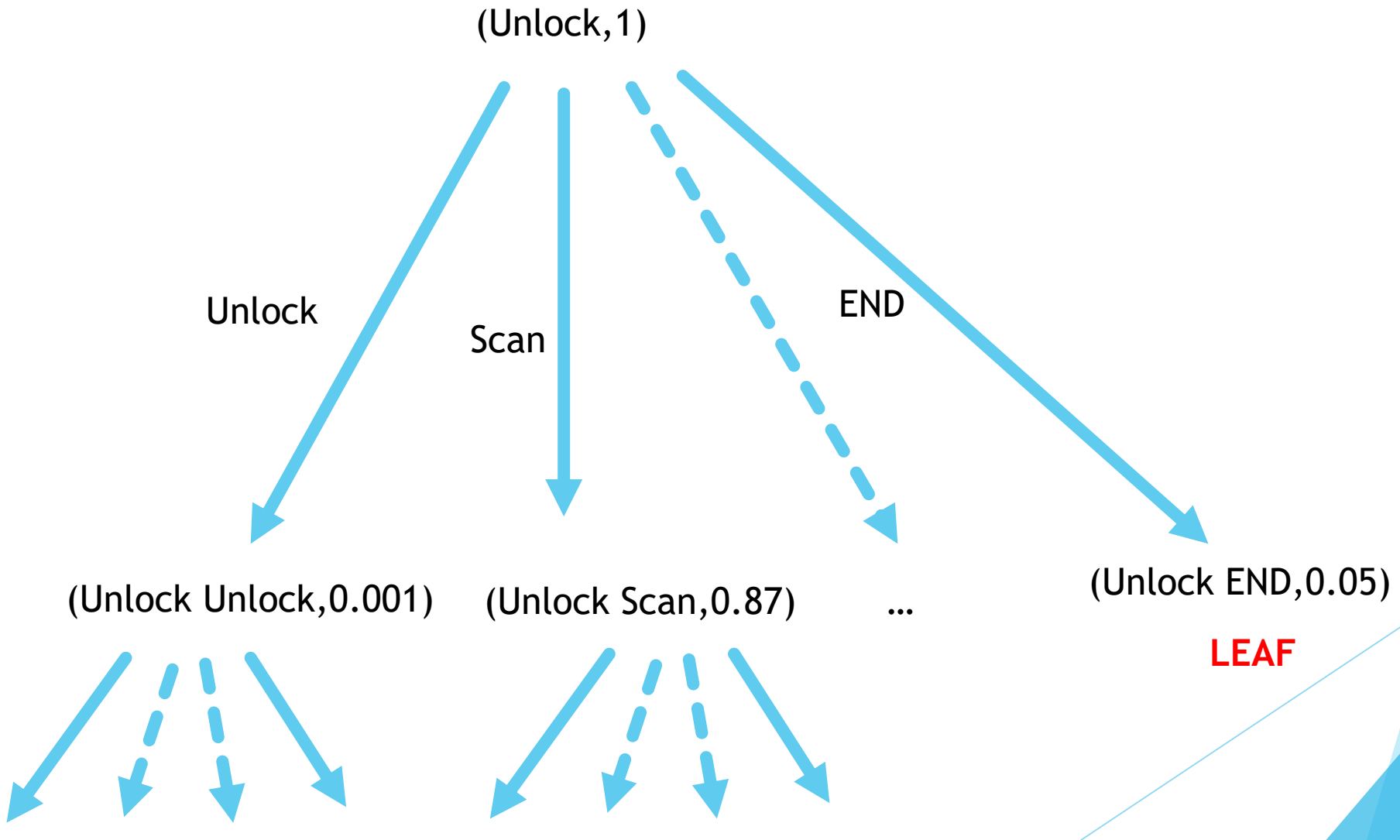
Classifier	Cluster3	Cluster4	Cluster5
Tree	0.961(0.051)	0.961(0.051)	0.991(0.051)
GBC	0.957(0.026)	0.961(0.051)	0.991(0.051)
RandForest	0.957(0.026)	0.966(0.035)	0.996(0.022)
AdaBoost	0.367(0.000)	0.374(0.006)	0.558(0.135)
NeuralNet	0.934(0.014)	0.947(0.037)	0.999(0.007)
Kneighbors	0.955(0.017)	0.960(0.042)	0.999(0.007)
NaiveBayes	0.856(0.022)	0.852(0.029)	0.827(0.000)
LinearsSVC	0.899(0.019)	0.852(0.029)	0.827(0.000)
LogReg	0.899(0.019)	0.852(0.029)	0.827(0.000)
Dummy	0.112(0.045)	0.117(0.052)	0.156(0.066)

F1 Score (Weighted average of precision and recall) for models learned from customer clusters 3-5

Generating Systematic Test suites

- We can generate all the most common sequences by unrolling our models
- The model has learned a function to map a trace prefix *tr* to probability distributions of the likely next events.
- Unrolling the model gives us a tree of *(tr,p)* where *tr* is a trace prefix and *p* the probability of that prefix.

Generating Systematic Test suites



Generating Systematic Test suites

21.90% u…..tap
16.52% u…….tap
10.61% u…….tao+cp
10.07% u…..tao+cp
05.23% u……tao+cp
03.72% u……tap
03.40% u…….tao++cp
02.56% u……tao++cp
02.11% u…….t…tap
01.61% u…..tao++cp
01.53% u……t.tap
01.25% u….tap
01.06% u……ttao+cp
82.57% of total behavior covered
Systematic test suite generated from the whole Scanner customer model, including all traces with probability greater than 1.0%

- Given an maximum trace lenght L and a minimum probability P, we can explore the tree via a depth-first recursive algorithm and extract the most common/representative traces
- Multiplying and summing the probabilities associated to our paths gives us the coverage of the traces generated regarding to our system

Application to two industry cases

Bus system and Supply chain

Bus system

- Web service for tracking school buses and students
- Events : GPS position of the bus, students swiping their ID cards upon entering or exiting the bus, drivers recordings absent students, etc :

3267 events from 15 buses and their frequencies :

SaveGPS '.'	2808.0
SNSCheckIn 'i'	133.0
SNSCheckOut 'o'	122.0
SNSMarkAbsentWithLocation 'A'	68.0
Login 'L'	30.0
GetSchoolManifestForRunType 'M'	30.0
ConfirmPreCheck 'P'	30.0
SNSBulkCheckOut 'O'	14.0
SNSBulkCheckIn 'I'	14.0
GetContacts 'C'	2.0
LoginOptions '?'	1.0

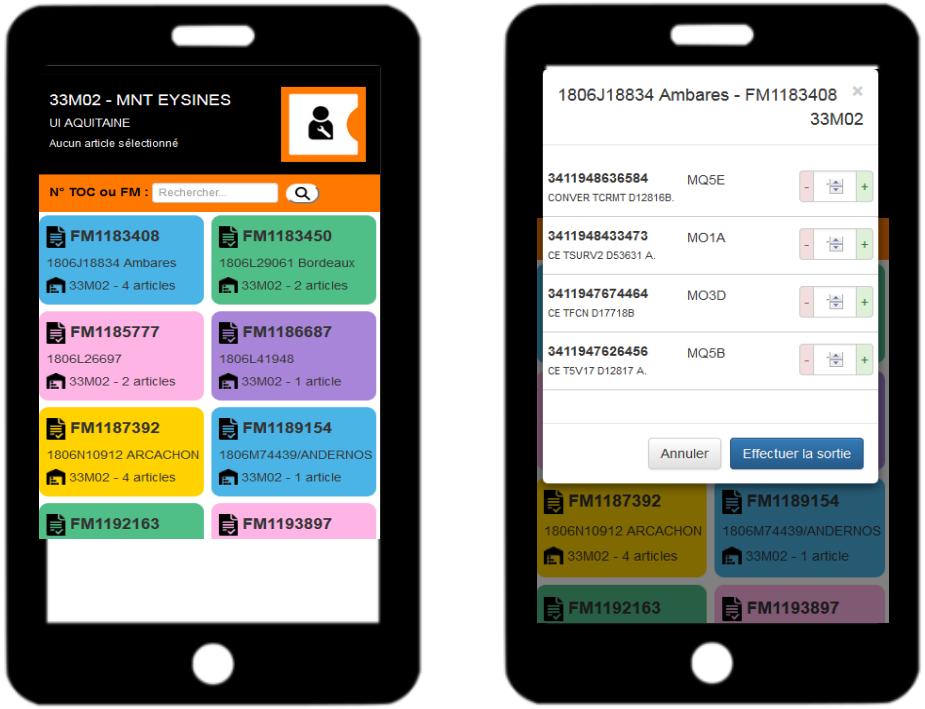
Bus system

```
 6.67% LP?M.ii.i.i.i.i.i.i.i.i.i.O.LPM.AA.I.(o.)^10
 5.56% LPM.i.i.i.i.i.i.i.O.LPM.I.o.o.o.o.o.o.o.
 4.44% LPMAAA.i.i.i.i.i.i.i.O.LPMAAA.A.A.I.(o.)^5
 4.44% LPMAAA.A.i.i.i.i.iO.LPM.AAAA.I.o.o.o.o.o.
 2.78% LPMA.C.i.i.i.i.i.i.i.i.i.O.LPMA.I.(o.)^6ooC.o.
 2.78% LPMA.i.i.i.i.i.i.i.i.O.LPM.I.o.o.o.o.o.o.
 1.67% LPMAA.A.i.i.i.i.i.i.i.O.LPMAAA.A.A.I.(o.)^5
 1.67% LPMAA.A.A.i.i.i.i.iO.LPM.AAAA.I.o.o.o.o.o.
 1.39% LPM.iiiA.i.i.i.i.i.i.i.i.i.O.LPMAI.(o.)^8ooo.o.
 1.25% LPMAA.i.i.ii.AAA.O.LPMAA.I.oo.ooooo.
 1.11% .LPM.i.i.i.i.i.i.O.LPM.I.o.o.o.o.o.o.
33.75% of total behavior covered
```

Systematic test cases generated by unrolling the probabilities tree

Supply chain

- Set of web services for managing maintenance equipment
- For each repair job, a list of required equipment is created by a remote operator
- Technicians use a mobile app to record when they collect and return the required equipment



PrepareOrder	'P'	374.0
DeliveredOrder	'D'	720.0
EquipmentReturn	'R'	227.0
CloseOrder	'V'	5.0
CreateCustomOrder	'N'	421.0
PrepareCustomOrder	'p'	468.0
StockConsistency	'-'	681.0
CancelOrder	'X'	2.0

2898 events and their frequencies coming from 437 sessions

Bus system

Id	Size	Description	Example
0	125	Creation of a custom order and then preparation and delivery.	NpD
1	85	Preparation and delivery of a regular order and then Stock Consistency call.	PD-
2	48	Same as cluster 1 but twice in a row.	NpDNpD
3	31	Equipment Return or Stock consistency call.	R
4	24	Sequences of cluster 2, then cluster 1.	PD-NpD
5	29	Same as cluster 2 but twice in a row.	NpDNpD

TABLE III

RESULT OF CLUSTERING FOR THE SUPPLY-CHAIN CASE STUDY.

Supply chain

```
24.26% NpD            15.23% PD-          7.55% NpDNpD
 3.89% R               2.19% PD-R         2.01% PD-PD-
 1.51% NpDNpDNpD       1.28% PD-NpD       1.07% NpDNpDR
 1.07% NpDR
60.05% total behavior covered
```

Systematic test cases generated by unrolling the probabilities tree

Future directions

- Test generation : learning of test data equivalence classes on test execution traces
- Provide a complete, open source toolbox

Thank you !