Project of

Fundamentals of Artificial Intelligence and Knowledge Representation

Module 3

Zoha Azimi Ourimi: zoha.azimiourimi@studio.unibo.it

Somayeh Shami: **somayeh.shami@studio.unibo.it**

Amir Hajiabadi: amir.hajiabadi@studio.unibo.it

Spring 2022

Abstract

Artificial intelligence and machine learning are making it possible for computers to diagnose some medical diseases more accurately than doctors. Such systems analyze millions of patient records and generalize to diagnose new patients. A key challenge is determining whether a patient’s symptoms are attributed to a known disease or other factors. Software testers face a similar problem when troubleshooting automation failures. They investigate questions like: Is a given failure due to a defect, environmental issue, or flaky test? Flaky tests exhibit both passing and failing results although neither the code nor test has changed. Maintaining flaky tests is costly, especially in large-scale software projects. In this paper, an approach that leverages Bayesian networks for classifying and predicting flaky tests is presented. This approach views the test flakiness problem as a disease by specifying its symptoms and possible causes.[1]

Model

Bayesian networks are probabilistic graphical models that measure the conditional dependence structure of a set of random variables based on the bayes theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayesian networks are graphical models that contain information about causal probability relationships between variables and are often used to aid in decision-making [2].

The causal probability relationships in a Bayesian network can be suggested by experts or updated using the Bayes theorem and new data being collected.

The inter-variable dependence structure is represented by nodes (variables) and directed arcs (conditional relationships) in the form of a directed acyclic (DAG):

- Each node corresponds to a random variable, which may be discrete or continuous
- A set of directed links connects pairs of nodes. If there is a link from node X to node Y, X is said to be parent of Y. The graph has no directed cycles
- Each node xi has a conditional probability distribution $P\left(xi|\left(Parents(xi)\right)\right)$ that quantifies the effect of the parents on the node.

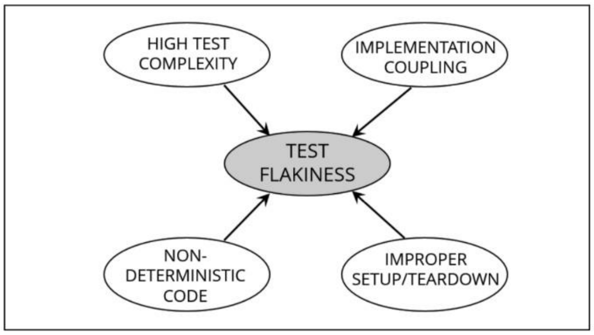
TEST FLAKINESS: AN AUTOMATION DISEASE

A flaky test may be defined as a test that exhibits both a passing and failing result with the same code [3], [4], and is therefore deemed unstable and unreliable. This approach views the test flakiness problem as a disease so that it can model its observable impacts as symptoms, and then determine any causal factors for those symptoms and the test flakiness disease itself. This section presents the overall approach to developing a prediction model for test flakiness and describes its disease-symptom relationships along with the proposed causal factors.

It should be mentioned that in our model, only a sub-net of the original network was considered. The involved variables are:

A - Causal Factors for Test Flakiness

Some of the factors that cause automated tests to be flaky are modeled in the below figure. These include:



- *High Test Complexity*: Large test scripts with many steps and assertions tend to be unstable. These types of automated tests often cover multiple scenarios, thereby violating the single responsibility principle [5]. Scripts that attempt to reuse code through inheritance and delegation may also exhibit unpredictable behavior.

- *Implementation Coupling*: Test scripts that are tightly coupled to the program implementation are also likely to be flaky. Examples of this include referencing concrete classes rather than interfaces from unit tests, or directly referencing web element selectors from within system-level UI test scripts. As the system under test evolves, such tests tend to break frequently.
- *Non-Deterministic Code*: Test scripts that contain conditional logic such as *if-then-else* and *try-catch* statements will take different paths depending on data and environmental conditions. This increases the probability that a test passes and fails even though the code has not changed. Another way that non-determinism can be introduced into test code is through explicit delays and random number generators.
- *Improper Setup/Teardown*: Test scripts that do not teardown data and environmental conditions properly will introduce instability into the pipeline. While such tests may not impact their own execution negatively, they have side effects that may cause subsequent tests to fail. Similarly, tests that do not properly establish preconditions are candidates for being classified as flaky tests.

B - SUPPORTING TEST METRICS

This section describes object-oriented test metrics [6] used to support data collection and discusses the rationale for collecting each metric.

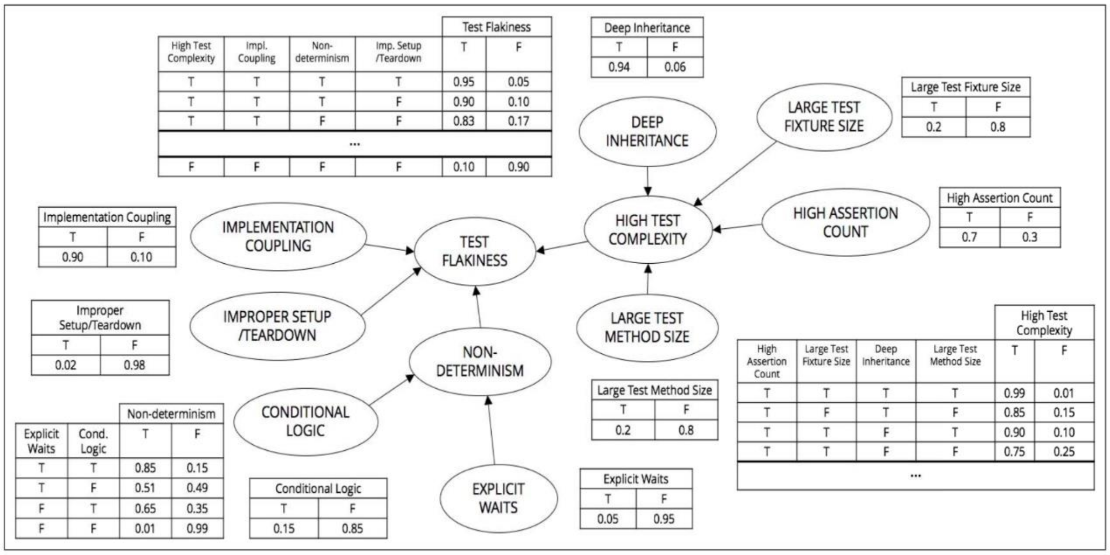
a. Complexity Metrics

- *Test Assertion Count*: the number of expressions that attempt to verify or confirm specific facts about the system under test. To analyze this metric at the test suite level, it is necessary to calculate the average assertion count across all test cases within each suite.
- *Test Class/Method Size*: the number of non-commented source lines of code associated with an automated test suite. To analyze this at the test suite level, the average method size across all tests should be collected.
- Depth of Inheritance Tree: the number of parent classes along the inheritance hierarchy, starting from the leaf test suite class and leading up to the root class.

b. Non-Determinism Metrics

- *Cyclomatic Complexity*: a measure of the number of linearly independent paths through a program [7]. Directly relates to the number of branches in the code.

- Explicit Wait Count: the number of statements in a test suite that cause forced delays to occur at runtime.



				Test Flakiness	
High Test Complexity	Impl. Coupling	Non-determinism	Imp. Setup /Teardown	T	F
T	T	T	T	0.95	0.05
T	T	T	F	0.90	0.10
T	T	F	F	0.83	0.17
...					
F	F	F	F	0.10	0.90

Deep Inheritance	
T	F
0.94	0.06

Large Test Fixture Size	
T	F
0.2	0.8

High Assertion Count	
T	F
0.7	0.3

Implementation Coupling	
T	F
0.90	0.10

Improper Setup/Teardown	
T	F
0.02	0.98

				High Test Complexity	
High Assertion Count	Large Test Fixture Size	Deep Inheritance	Large Test Method Size	T	F
T	T	T	T	0.99	0.01
T	F	T	F	0.85	0.15
T	T	F	T	0.90	0.10
T	T	F	F	0.75	0.25
...					

Large Test Method Size	
T	F
0.2	0.8

		Non-determinism	
Explicit Waits	Cond. Logic	T	F
T	T	0.85	0.15
T	F	0.51	0.49
F	T	0.65	0.35
F	F	0.01	0.99

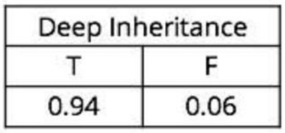
Conditional Logic	
T	F
0.15	0.85

Explicit Waits	
T	F
0.05	0.95

Conditional probability distribution tables

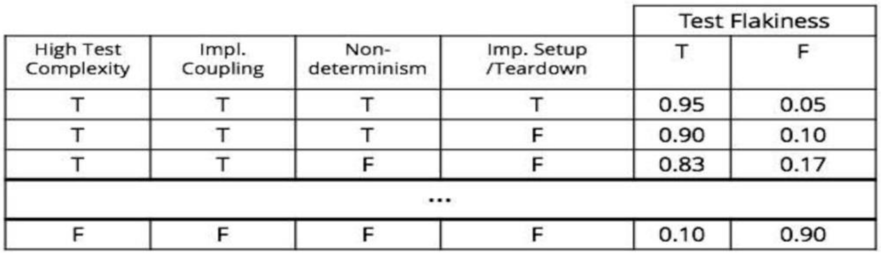
Conditional probability distributions (CPDs) are a form of tabular representation suitable for discrete variables. Each row in a CPD contains the conditional probability of each node value for a conditioning case. A conditioning case is just a possible combination of value for the parent nodes.

The following tables are the CPDs for the nodes Deep Inheritance



Deep Inheritance	
T	F
0.94	0.06

The following tables are the CPDs for the nodes Test Flakiness and High-Test Complexity, Implementing Coupling, Non-determinism and Implementation. setup/Teardown



				Test Flakiness	
High Test Complexity	Impl. Coupling	Non-determinism	Imp. Setup /Teardown	T	F
T	T	T	T	0.95	0.05
T	T	T	F	0.90	0.10
T	T	F	F	0.83	0.17
...					
F	F	F	F	0.10	0.90

It should be mentioned that due to lack of the full data for conditional probability distribution tables for two variables "test flakiness" and "non-determinism", we assigned the missing values using the pattern of available data, considering the effect of each variable on the result according to the paper.

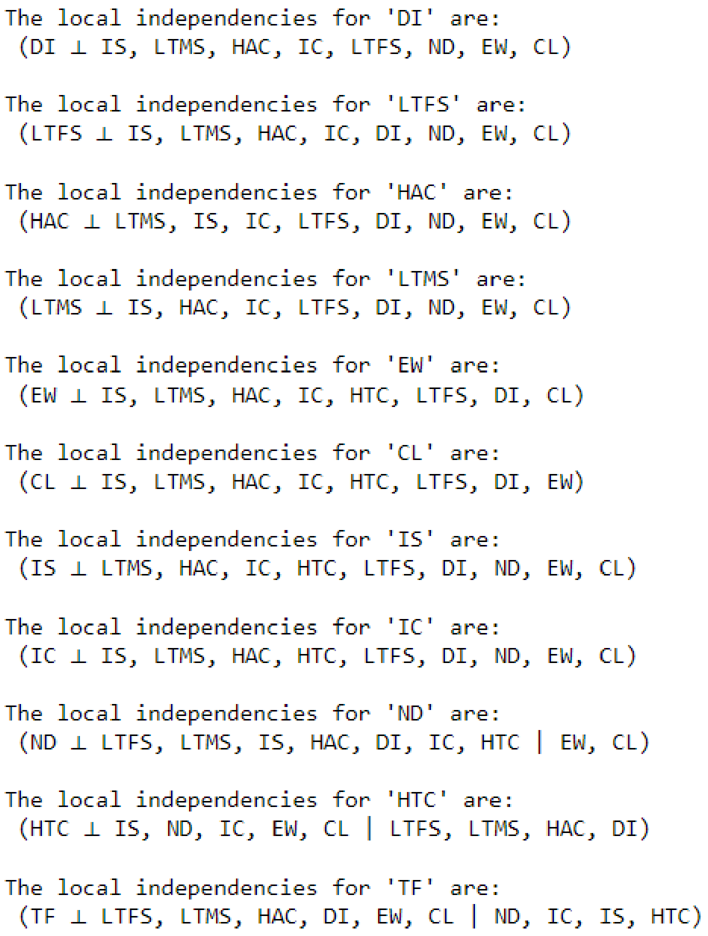
Also, following abbreviations are used in the following sections of this report:

Variable	Abbreviated as
Test flakiness	TF
High test complexity	HTC
Non-determinism	ND
Image setup/ teardown	IS
Implementation coupling	IC
Explicit waits	EW
Conditional logic	CL
Large test method size	LTMS
Large test fixture size	LTFS
Deep inheritance	DI
High assertion count	HAC

Investigating the model

Independence

In our network, there can be made 9016 valid independence assertions, with respect to all possible given evidence. The main local independencies are:



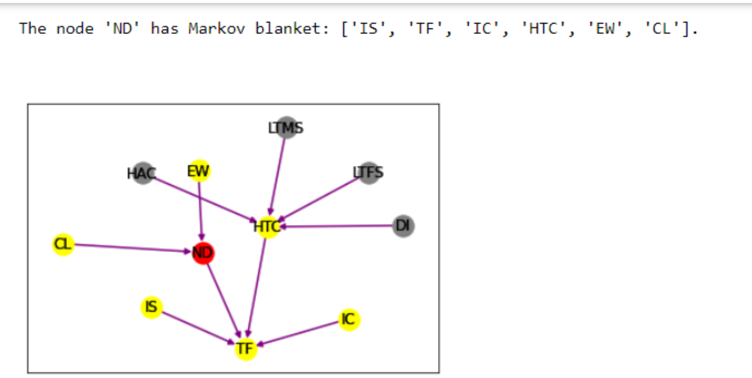
```
The local independencies for 'DI' are:
 (DI ⊥ IS, LTMS, HAC, IC, LTFS, ND, EW, CL)

The local independencies for 'LTFS' are:
 (LTFS ⊥ IS, LTMS, HAC, IC, DI, ND, EW, CL)

The local independencies for 'HAC' are:
 (HAC ⊥ LTMS, IS, IC, LTFS, DI, ND, EW, CL)

The local independencies for 'LTMS' are:
 (LTMS ⊥ IS, HAC, IC, LTFS, DI, ND, EW, CL)

The local independencies for 'EW' are:
 (EW ⊥ IS, LTMS, HAC, IC, HTC, LTFS, DI, CL)

The local independencies for 'CL' are:
 (CL ⊥ IS, LTMS, HAC, IC, HTC, LTFS, DI, EW)

The local independencies for 'IS' are:
 (IS ⊥ LTMS, HAC, IC, HTC, LTFS, DI, ND, EW, CL)

The local independencies for 'IC' are:
 (IC ⊥ IS, LTMS, HAC, HTC, LTFS, DI, ND, EW, CL)

The local independencies for 'ND' are:
 (ND ⊥ LTFS, LTMS, IS, HAC, DI, IC, HTC | EW, CL)

The local independencies for 'HTC' are:
 (HTC ⊥ IS, ND, IC, EW, CL | LTFS, LTMS, HAC, DI)

The local independencies for 'TF' are:
 (TF ⊥ LTFS, LTMS, HAC, DI, EW, CL | ND, IC, IS, HTC)
```

For example, for the variable TF, we see that given ND, IC, IS and HTC, which are the Markov blanket, it will be independent of other variables.

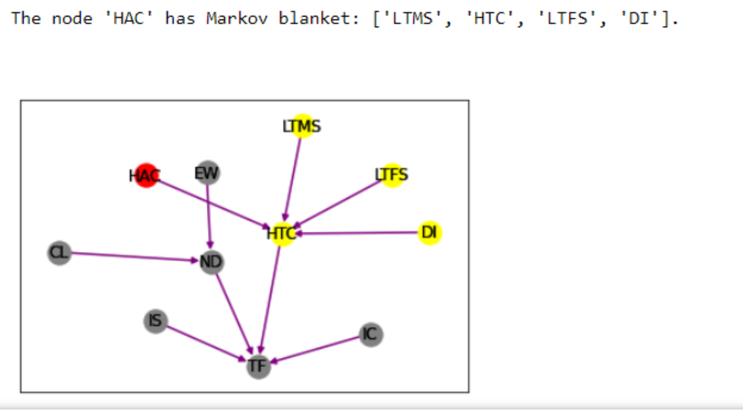
Markov Blanket

The Markov blanket of a given node (in red) consists in the set (in yellow) of:

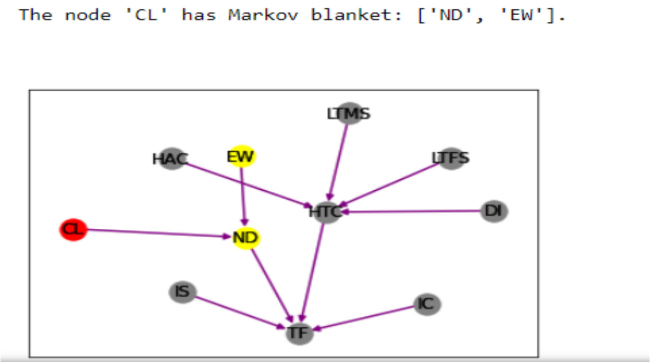
- Its parents
- Its children
- Its children ‘s other parents
- Each node is conditionally independent of all others, given its Markov blanket. The followings are the Markov blankets for the nodes HAC and ND and CL.



```
The node 'ND' has Markov blanket: ['IS', 'TF', 'IC', 'HTC', 'EW', 'CL'].
```



```
The node 'HAC' has Markov blanket: ['LTMS', 'HTC', 'LTFS', 'DI'].
```



```
The node 'CL' has Markov blanket: ['ND', 'EW'].
```

Queries

Exact inference with variable elimination

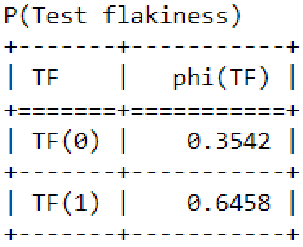
The variable elimination is one of the most widespread methods for exact inference. It is an improvement of the inference by enumeration, where the repeated sub-expression of the query is evaluated only once (dynamic programming).

Different orderings of the variables cause different intermediate factors to be generated during the calculation, but every choice yields a valid algorithm.

In general, the time and space requirements of variable elimination are dominated by the size of the largest factor constructed during the operation of the algorithm. This in turn is determined by the order of elimination of variables and by the structure of the network. It is interactable to determine the optimal ordering, but several good heuristics are available.

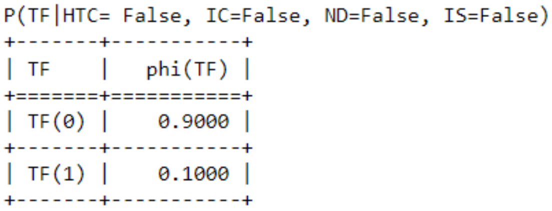
Some interesting queries

- The probability of having a Flaky test:



```
P(Test flakiness)
+-------+-----------+
| TF    |   phi(TF) |
+=======+===========+
| TF(0) |    0.3542 |
+-------+-----------+
| TF(1) |    0.6458 |
+-------+-----------+
```

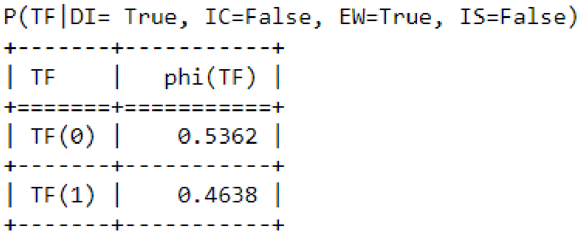
The probability of having flaky test given that HTC, IC, ND and IS are false:



```
P(TF|HTC= False, IC=False, ND=False, IS=False)
+-------+-----------+
| TF    |   phi(TF) |
+=======+===========+
| TF(0) |    0.9000 |
+-------+-----------+
| TF(1) |    0.1000 |
+-------+-----------+
```

It is expected to have such a result since HTC, IC, ND and IS are causal factors of having a flaky test and their absence must lower the probability of test flakiness occurrence.

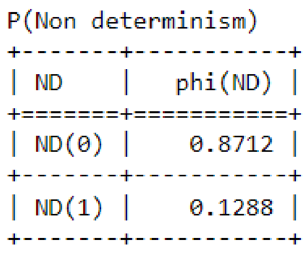
- The probability of having flaky test given that DI, EW is true and IC, IS are false (is higher than previous query):



```
P(TF|DI= True, IC=False, EW=True, IS=False)
+-------+-----------+
| TF    |   phi(TF) |
+=======+===========+
| TF(0) |    0.5362 |
+-------+-----------+
| TF(1) |    0.4638 |
+-------+-----------+
```

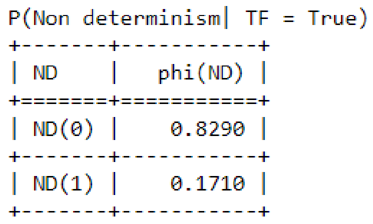
As expected, the probability increases when DI and EW are given to be True, since they effect the variables HTC and ND, the causal factors of TF.

- The probability of non-determinism:



```
P(Non determinism)
+-------+-----------+
| ND    |   phi(ND) |
+=======+===========+
| ND(0) |    0.8712 |
+-------+-----------+
| ND(1) |    0.1288 |
+-------+-----------+
```

- The probability of non-determinism given TF true:



```
P(Non determinism| TF = True)
+-------+-----------+
| ND    |   phi(ND) |
+=======+===========+
| ND(0) |    0.8290 |
+-------+-----------+
| ND(1) |    0.1710 |
+-------+-----------+
```

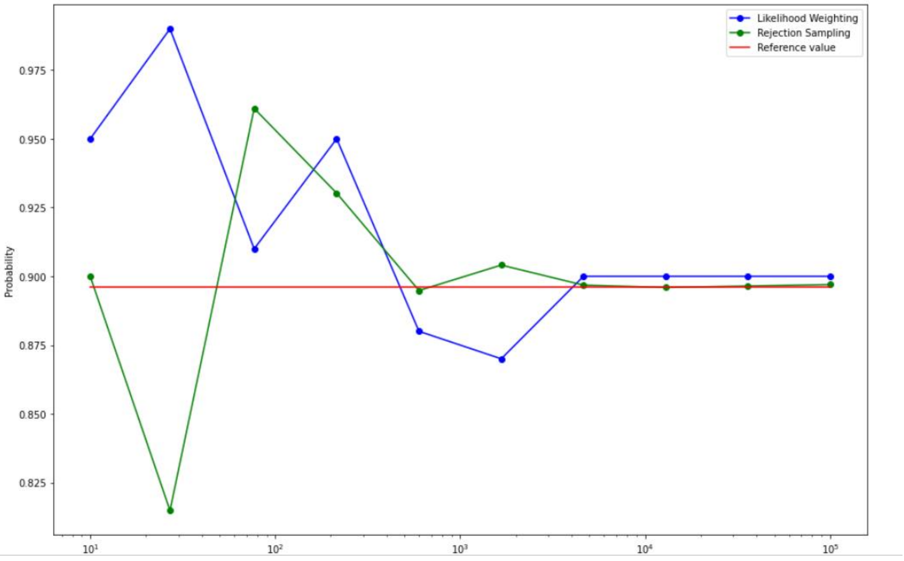
Again, given TF to be true, as expected, the probability of ND will increase compared to the probability of ND without any evidence.

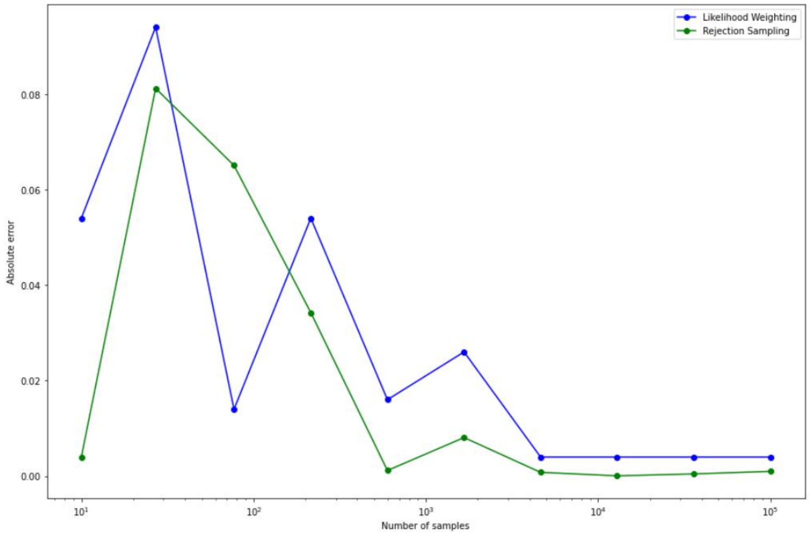
Approximate inference

Exact inference is carried out with posterior probabilities that are not always tractable. Approximate inference methods address this issue by sampling from the un-tractable posterior (stochastic methods) or by approximating the posterior with a tractable distribution (deterministic methods).

To study this way of inference, we used two inference method:

- **Rejection sampling:** in a general method for producing samples from hard-to-sample distribution given an easy-to-sample distribution. In its simplest form, it can be used to compute conditional probabilities. The algorithm first, generates samples from the prior distribution specified by the network. Then, it rejects all those that do not match the evidence. Finally, the estimate is obtained by counting how often the evidence occurs in the remaining samples.
- **Likelihood inference:** it avoids the inefficiency of rejection sampling by generating only events that are consistent with the evidence. The algorithm fixes the values for the evidence variables and samples only non-evidence variables. This guarantees that each event generated is consisted with the evidence. Nevertheless, not all events are equal. Before tallying the counts in the distribution of the query variable, each event is weighted by the likelihood that the event accords to the evidence, as measured by the product of the conditional probabilities for each evidence variable, given its parents. Intuitively, events in which the actual evidence appears unlikely should be given less weight.





Figures above show the probability of TF given ND and HTC in 3 different cases of exact inference, approximate inference with rejection sampling and approximate inference with likelihood sampling.

The graphs show that, for our model, the approximate methods require a lot of computation with respect to the exact inference (10^5 samples), to converge. Despite this computational effort, the result is good because we can see that both approximating methods, converge well towards the reference value, that is computed by an exact inference.

Conclusion

The case study of the paper with the goal of improving the efficiency of the testing process at Ultimate Software, provided evidence that the approach presented in the paper is feasible, however limited and it can be evaluated more.

References

[1] T. M. King, D. Santiago, J. Philips, P. J. Clarke. "Towards a Bayesian Network Model for Predicting Flaky Automated Tests", IEEE International Conference on Software Quality, Reliability and Security, 2018. Companion

[2] Shen Liu et al. Computational and Statistical Methods for Analyzing Big Data with Application. English. Netherlands: Elsevier, 2016. ISBN:9780128037324. DOI: 10. 1016/C2015-0-00198-8.

[3] Z. Gao. Quantifying Flakiness and Minimizing its Effects on Software Testing. PhD thesis, University of Maryland, 2017.

[4] JetBrains. TeamCity - A Hassle-free CI and CD Server, April 2018. https://www.jetbrains.com/teamcity/.

[5] Martin, Robert C. Clean Coder - Getting a SOLID start, February 2009. http://sites.google.come/site/ unclebobconsultingl/ getting- a- solid- start.

[6] S. R. Chidamber and C. F. Kemerer. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6):476–493, Jun 1994.

[7] T. J. McCabe. A complexity measure. IEEE Transactions on software Engineering, (4):308–320, 1976.