RN/11/09
March 14, 2011

Research
Note

FLINT: Fault Localisation using Information Theory

S. Yoo, M. Harman & D. Clark

Telephone: +44 (0)20 3108 5032
Fax: +44 (0)171 387 1397
Electronic Mail: {S.Yoo, M.Harman, D.Clark}@cs.ucl.ac.uk
URL: http://www.cs.ucl.ac.uk/staff/S.Yoo/,
http://www.cs.ucl.ac.uk/staff/M.Harman/,
http://www.cs.ucl.ac.uk/staff/D.Clark/

Abstract

Test case prioritisation techniques aim to maximise the chance of fault detection as early in testing as possible. This is most commonly achieved by prioritising the tests according to a surrogate measure that is thought to correspond to fault detection capabilities, such as code coverage. However, once the prioritised test suite indeed detects a fault, the original prioritisation may become obsolete. Rather, from the point of the first fault detection, the aim of the prioritisation should be that it should maximise the chance of locating the detected fault. This paper introduced a novel dynamic test prioritisation technique that is based on Shannon's entropy. Fault localisation is formulated as a process of decreasing the entropy calculated over the test information. The dynamic test case prioritisation uses both the coverage information from the previous testing and the results from the current testing and selects the next test that is most likely to reduce the entropy of information regarding the locality of the fault, maximising the chance of identifying the location of the fault whenever the testing is terminated.

Keywords

Information Theory, Fault Localisation, Test Case Prioritisation

Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK

1 Introduction

Fault localization can be used to prioritize the statements of a program according to the likelihood that each contains a fault that causes a known failure [16, 20]. This provides decision support to the engineer which may help to reduce the time taken to find the fault [15]. Fault localization is usually based on a concept of suspiciousness: we assign a degree of suspiciousness to each statement that assesses the likelihood that it contains the fault, based on the evidence from testing.

In this paper we chose to incorporate the widely used Tarantula suspiciousness metric [15, 16] into our approach. However, the choice of suspiciousness measurement is a parameter to our approach, so we can easily incorporate metrics from other fault localization work [10, 19, 20].

Previous work on fault localization tacitly assumes that all test cases contribute positively to the localization of a fault. However, some test cases execute a faulty statement without leading to failure. Such test cases cause the true faulty statement to appear less suspicious. The information they provide can thus reduce the likelihood that the engineer will find the faulty statement early. Borrowing from the famous aphorism of Fred Brooks [6], we might say that

"adding tests to a late localization may make it later".

In order to address this problem we introduce FLINT: Fault Localization using INformation Theory. We use Shannon's Information Theory [22] to define an entropy measure for test cases, such that the next test case in an ordering is the one that maximally reduces fault locality entropy. This is the first time that information theory has been incorporated into fault localization. All fault localizaton approaches implicitly rely on some form of information to reduce the inherent uncertainty surrounding fault location. This makes Shannon's mathematical theory of information a natural choice; one that we formalise in this paper. From a more practical point of view, our FLINT approach allows us to maximise the chance of localizing the fault, even when we are only able to consider an initial prefix of test case ordering.

We report the results of an empirical study of an implementation of FLINT on five different releases of four software systems for which test suites and fault information are available. Our results show that information theoretic test ordering can outperform coverage based ordering for early fault localization with statistical significance. Our study also reveal examples in which the theoretical observation "adding tests to a late localization may make it later" is borne out in practice, providing a further motivation for the incorporation of information theory into fault localization.

In our approach statements of the program are ordered according to their suspiciousness, as is common with other work on fault localization. However, test cases are also ordered according to their ability to reduce fault localization entropy. The engineer can thus consider first the most suspicious statements, but also can consider first the test cases that reveal most information about the likely faultiness of each statement considered.

Regression testing and fault localization are naturally complementary: we test to see if change has introduced a fault and, if we discover that it has, we switch from regression testing to fault localization as a first step towards fault fixing. Having fixed the fault we switch back to regression testing and so the cycle continues. This 'test–find–fix' cycle is familiar to many software engineers, yet surprisingly, it fails to find a compelling counterpart in the literature. FLINT allows us to unite the two related, but hitherto largely disjoint sets of literature on test case prioritization and fault localization.

Regression test prioritization is used to order statements according to early achievement of some testing goal such as coverage [14, 18, 21, 24]. Should any test lead to a failure, we immediately switch from regression testing to fault localization. An unstated assumption in previous work on fault localization is that we should use all available test cases to localize a fault. FLINT challenges this assumption and this is important for this 'test–find–fix' cycle, because when we switch from regression testing to fault localization not all test cases will yet have been executed, thereby raising the natural question:

> "Having found a fault, what is the best order in which to rank the remaining test cases to maximise early localization of the fault?"

The FLINT approach addresses precisely this question. For those test cases already executed at the point we switch to localization we shall have reliable coverage. However, for those that remain to be executed, we shall have to make do with coverage information available from the previous version of the software. Of course, we could simply let the regression testing run on to completion, but for smaller changes complete retesting may be impractical. Complete retesting times of up to seven weeks have been reported in the literature [21]. Clearly, it would therefore be unreasonable to always expect the engineer to wait for regression completion before fault localization can commence.

Our second empirical study evaluates the performance of the FLINT approach for the 'test–find–fix' cycle. The results indicate that even in this information-impoverished environment, the FLINT approach outperforms traditional coverage-based test case prioritization, localizing faulty statements statistically significantly sooner in all programs studied. This provides a further motivation for the information theoretic approach to fault localization that we advocate. We may summarise the primary contributions of the paper as follows:

1. The paper is the first to use of Information Theory to capture the entropy of the locality of faults. Shannon entropy provides a quantitative theoretical foundation on which to build a new approach to fault localization in which both statements and test cases are prioritized. Statements are ordered by suspiciousness, while test cases are ordered by the degree to which they reduce the entropy inherent in fault localization.
2. The paper presents the results of an empirical study that demonstrates that information theoretic ordering outperforms coverage-based test case prioritization.
3. The paper presents results from a further empirical study that provides evidence to support the claim that information theoretic ordering copes well with imperfect, partial and noisy information. This makes the approach applicable after code changes have degraded existing test coverage information.

The rest of the paper is organised as follows: Section 2 explains the underlying concepts of the suspiciousness metric and test case prioritisation as well as presenting a motivating example for FLINT. Section 3 presents the theoretical foundation of FLINT approach and sets out the research questions. Section 4 describes the algorithms for FLINT. Section 5 discusses the experimental setup of the empirical study, the results of which are analysed in Section 6. Section 7 present related work and Section 8 concludes.

2 Background

2.1 Test Case Prioritisation

Test case prioritisation concerns ordering test cases for early maximisation of some desirable properties, such as the rate of fault detection [24]. It seeks to find the optimal permutation of the sequence of test cases. It does not involve selection of test cases, and assumes that all the test cases may be executed in the order of the permutation it produces, but that testing may be terminated at some arbitrary point during the testing process. More formally, the prioritisation problem is defined as follows:

Definition 1 ***Test Case Prioritisation Problem***

Given: *a test suite, T, the set of permutations of T, PT, and a function from PT to real numbers, $f : PT \rightarrow \mathbb{R}$.*

Problem: *to find $T' \in PT$ such that $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.*

The ideal choice of the priority function, f, would be one that would result in an ordering of tests with the maximum rate of fault detection. Since this information is unavailable before the testing is finished, various surrogates including code coverage [14, 21], test execution history [17] and expert knowledge [23, 25] have been studied.

2.2 Fault Localisation Metrics

Fault location techniques aim to reduce the cost of debugging by automating the process of searching for the location of the fault in the program. A widely studied approach to fault localisation is to assign to each structural element in the program a *suspiciousness* value that corresponds to the relative likelihood of the element containing the fault [1, 15, 19]. For example, the Tarantula suspiciousness metric [15] for a statement s in a program is calculated as follows:

$$\text{Tarantula metric } \tau(s) = \frac{\frac{fail(s)}{totalfail}}{\frac{pass(s)}{totalpass} + \frac{fail(s)}{totalfail}} \quad (1)$$

In Equation 1, $fail(s)$ and $pass(s)$ represent the number of times the statement s was executed by failing and passing tests, respectively, whereas $totalfail$ and $totalpass$ represent the number of failing and passing tests.

The highest possible value for τ is 1 and the lowest is 0. If a statement s is executed by all tests, regardless of their results, it gets assigned $\tau = 0.5$. A faulty statement s' gets assigned $\tau = 1$ if and only if all failing tests and none of the passing tests executes s'. However, it is possible that some statements other than s' gets a higher τ value than s'. Suppose that s' causes a failure only for certain input values, whereas an error handling routine s'' is executed whenever s' fails: s'' will get assigned $\tau = 1$, whereas s' might get assigned τ less than 1 depending on the test input.

2.3 Prioritising Tests for Fault Localisation

Existing work on fault localisation treated the calculation of suspiciousness metrics as a *post hoc* procedure. That is, fault localisation was attempted only after the entire test suite was executed. However, this contradicts the assumptions behind test case prioritisation, i.e. that there may not be enough time to execute the entire test suite.

Suppose that the tester encounters a failing test while executing a test suite prioritised for maximum fault detection capability. We argue that, after the initial failure, different tests contribute different amounts of information regarding the location of the faulty structural element. It follows that, after the initial failure, the tester should choose a test that would provide the most information as the next test case whenever possible, followed by other tests in the order of decreasing amount of information provided.

Consider the motivating example in Table 1. Test t_1 to t_4 is prioritised based on the structural coverage following the *additional* approach with resets [14]. The dots (•) show the coverage relation: for example, structural element s_1 is covered by test t_1 and t_3. The prioritised test suite detects the first fault with t_2, which covers the faulty element s_7. Suppose that there is only time to execute one additional test: the next two columns show what the final suspiciousness metric would look like if t_3 or t_4 is chosen to be executed. According to the coverage-based prioritisation, the next test is t_3 and the faulty element will get assigned the suspiciousness of 0.67. However, this is misleading as s_6 and s_8 are assigned with higher suspiciousness values. On the other hand, if t_4 is executed, the faulty element is assigned the suspiciousness of 1.0 along with other elements, which would be a more precise result. This shows that the choice of the next test case can affect the accuracy of the suspiciousness metric if the testing is terminated at an arbitrary point.

In reality, it is impossible to predict whether a test would pass or fail. Therefore, it is also impossible to make the ideal choice for fault location. However, it is possible to formulate a probabilistic approximation

Structural Elements	Test t_1	Test t_2	Test t_3	Tarantula Metric(τ)	Test t_4	Tarantula Metric(τ)
s_1	•		•	0.00		0.00
s_2	•		•	0.00		0.00
s_3	•		•	0.00		0.00
s_4	•			0.00		0.00
s_5	•		•	0.00		0.00
s_6		•		1.00	•	1.00
s_7 (faulty)		•	•	0.67	•	1.00
s_8		•		1.00	•	1.00
s_9	•	•		0.67	•	0.50
Result	P	F	P	-	F	-

Table 1: Motivating Example: coverage-based prioritisation would execute t_3 after the first failure (t_2), resulting in sub-optimal suspiciousness metric values. However, if t_4 is executed after the first failure, the faulty s_7 will get assigned the optimal suspiciousness value.

that can be used as a surrogate, much in the same way as test case prioritisation techniques use structural coverage as a surrogate for the measure of fault detection capability. It is for this that we turn to Information Theory.

3 Fault Localisation & Entropy

This section presents the formulation of fault localisation as an entropy reduction process and outlines the research questions.

3.1 Problem Formulation

3.1.1 Assumptions & Basic Notations

Let $S = \{s_1, \ldots, s_m\}$ be the set of structural elements in the System Under Test (SUT); let $T = \{t_1, \ldots, t_n\}$ be the test suite with n tests. A single element in S contains the fault. Let $C : T \rightarrow 2^S$ be the mapping from tests to executed structural elements, i.e.:

$$C(t) = \{s \in S | t \text{ covers } s \text{ when executed}\}$$

Finally, let $F(t)$ be a boolean statement that says test t has failed. Similarly, let $B(s)$ be a boolean statement that says structural element s contains a fault. We will make the following assumptions for our approximation:

1. The results from all tests in T are deterministic, i.e. $\forall t \in T : F(t) \vee \neg F(t)$.
2. The suspiciousness metric is *competent* and does reflect the likelihood of faultiness, i.e. $\mathbf{P}(B(s)) \sim \tau(s)$.
3. The mapping between tests and structural elements, C, is known.

The first assumption underpins most existing work for software testing. The second assumption is supported by empirical evidence in the existing work [1, 15], the findings of which the paper replicates. Regarding the third assumption, the empirical study in the paper considers both the case: when it holds and when it does not. The assumption about the knowledge of coverage information may not be realistic in certain cases. However, when the exact information C is not known, it is possible to replace C with information from the previous iteration of testing, similar to the way in which test case prioritisation techniques use the coverage information from the previous iteration of testing.

Now we describe the situation in which the ith test fails during testing. Without losing generality, let T_{i-1} be the set of $i-1$ tests, $\{t_1, \ldots, t_{i-1}\}$, that have passed; let t_i be the first failing test. For the sake of brevity, let TP_i and TF_i be the total number of passing/failing tests, respectively, after executing the tests in T_i. Similarly, let $CP_i(s_j)$ and $CF_i(s_j)$ be the times s_j has been covered by passing/failing tests, respectively, after executing the tests in T_i.

3.1.2 Entropy of Fault Locality

Given a set of tests at least one of which fails, it is possible to calculate the suspiciousness of each statement based on the tests executed up to and including the point of the first failure. Given a set of tests $T_i = T_{i-1} \cup \{t_i\}$, let $\tau(s|T_i)$ denote the suspiciousness of s calculated using the tests in T_i. Based on the assumption 2, the probability that statement s_j contains the fault, based on the information observed with T_i, is calculated as the normalised suspiciousness metric for s_j:

$$\mathbf{P}_{T_i}(B(s_j)) = \frac{\tau(s_j|T_i)}{\sum_{j=1}^{n} \tau(s_j|T_i)} \tag{2}$$

Shannon's entropy regarding the locality of the fault can now be defined as follows:

$$\mathbf{H}_{T_i}(S) = -\sum_{j=1}^{n} \mathbf{P}_{T_i}(B(s)) \cdot \log \mathbf{P}_{T_i}(B(s)) \tag{3}$$

Ideally, fault localisation is complete when $\mathbf{H}(S)$ reaches 0: the probability $\mathbf{P}(B(s'))$ will be 1 for the fault statement s' and 0 for the remaining. Since we use $\tau(s)$ as a surrogate for the real probability distribution, our aim is to minimise $\mathbf{H}(S)$ as much as possible. This means not only increasing the suspiciousness of the faulty statement, but also decreasing the suspiciousness of the non-faulty statements.

When locating a fault that can be detected deterministically (Assumption 1), it should be noted that the entropy of fault locality, calculated following Equation 3, is identical for the same set of tests, i.e. $T = T' \rightarrow \mathbf{H}_{T_{i+1}}(S) = \mathbf{H}_{T'_{i+1}}(S)$. That is, the same set of tests yields the same amount of information regarding the locality of the fault. The aim of FLINT is not, and cannot be, to increase the amount of information; rather, it is to order tests so that the maximum information is extracted as early as possible. It follows that the next test to execute, t_{i+1}, should be the one that yields the smallest $\mathbf{H}_{T_{i+1}}(S)$.

3.1.3 Entropy Lookahead

To calculate $\mathbf{H}_{T_{i+1}}(S)$, $\mathbf{P}_{T_{i+1}}(B(s_j))$ needs to be approximated. Since it is not possible to predict whether t_{i+1} will pass or fail, we use conditional probability to express both cases:

$$\begin{aligned} \mathbf{P}_{T_{i+1}}(B(s_j)) = \quad & \mathbf{P}_{T_{i+1}}(B(s_j)|F(t_{i+1})) \cdot \mathbf{P}_{T_{i+1}}(F(t_{i+1})) + \\ & \mathbf{P}_{T_{i+1}}(B(s_j)|\neg F(t_{i+1})) \cdot \mathbf{P}_{T_{i+1}}(\neg F(t_{i+1})) \end{aligned} \tag{4}$$

The conditional probability $\mathbf{P}_{T_{i+1}}(F(t_{i+1})|B(s_j))$ can be approximated by following Equation 2 with assumptions about the outcome of t_{i+1}: we simply consider two separate cases (t_{i+1} passes or fails) and calculate the lookahead suspiciousness metric accordingly.

The remaining unknown terms are essentially probabilities that the next test either passes or fails. Instead of using arbitrarily fixed vlaues, we use the observed feedback from the execution of tests in T_i as follows:

$$\mathbf{P}_{T_{i+1}}(F(t_{i+1})) \approx \frac{TF_i}{TP_i + TF_i} \tag{5}$$

$$\mathbf{P}_{T_{i+1}}(\neg F(t_{i+1})) \approx \frac{TP_i}{TP_i + TF_i} \tag{6}$$

Using the lookahead suspiciousness, Equation 5 and 6, it is possible to calculate Equation 4, i.e. the lookahead probability. Once normalised, the lookahead probability enables the calculation of the lookahead entropy that is expected from the execution of each candidate test case for t_{i+1}. For better fault localisation after the detection of the first failing test, the tester should select the next test case that is expected to yield the lowest entropy by the approximation.

3.2 Research Questions

The "Precision" study considers the case when C, i.e. the coverage information of the each test, is known. The aim is to investigate what FLINT is capable of when C is known: this corresponds to the use case when the tester wants guidance for debugging by ranking tests *post hoc*, i.e. after executing all the tests, in the order of the amount of information they reveal regarding the locality of the fault.

RQ1. Effectiveness: Does FLINT increase the suspiciousness of the faulty statement during testing? If so, by how much?

RQ2. Efficiency: If FLINT successfully increase the suspiciousness of the faulty statement, does this result in reduction in number of statements to be inspected before the tester encounters the faulty statement?

RQ1 is answered by observing the suspiciousness metric of the faulty statement during the execution of the test suite in two different orders: coverage-based prioritisation and entropy-based prioritisation. **RQ2** is answered by analysing the percentage of the number of statements that the tester has to investigate, following the suspiciousness ranking, until the faulty statement is encountered.

The next set of research questions considers "Robustness" of the proposed technique: we assume the situation when C, the coverage relation between tests and statements, is not known and, therefore, has to be replaced by the coverage information from the previous version. This corresponds to the use case when the tester wants to maximise the efficiency of fault localisation by executing tests in the order of the amount of information they reveal.

RQ3. Robustness: Does the use of coverage information from the previous iteration of testing affect the effectiveness of FLINT? If so, by how much?

RQ4. Correlation: Does the result from the previous version of SUt correlate with the result from the current version?

RQ3 is answered by applying FLINT using coverage information from the previous iteration of testing and analysing the results. **RQ4** is answered by analysing the statistical correlation between results from two consecutive versions of SUT: this enables the tester to estimate the potential reduction FLINT can introduce to the testing of SUT.

4 Algorithms

4.1 Entropy Lookahead Algorithm

To keep the pseudo-code concise, let us assume that the counter functions described in Section 3.1.1, $TP, TF, CP_i : S \rightarrow \mathbf{N}$ and $CF_i : S \rightarrow \mathbf{N}$, as well as the coverage relation, C, remain available throughout Algorithm 1, 2 and 3. The variable i denotes, whenever present, that the algorithm is seeking to find the i-th test.

Algorithm 1 contains the lookahead algorithm for entropy. The loop in Line (1) calculates the lookahead suspiciousness for each statement, s_j: $t_{p,j}$ is the lookahead suspiciousness of s_j when the candidate test t is expected to pass and $t_{f,j}$ when t is expected to fail. The lookahead for each case is achieved by providing different input values to Algorithm 2.

The loop in Line (10) uses Equation 4 in Section 3.1.3 in order to convert the lookahead suspiciousness values into lookahead probabilities., which is normalised using P_{sum}. Finally, Line (14) normalises the lookahead probabilities using P_{sum} and summarises them into the entropy H, following Equation 3.

Algorithm 1: Entropy Lookahead Algorithm
EL(t)
Input: A candidate test, t
Output: Lookahead entropy, H
(1) **foreach** $s_j \in S$
(2) $\quad c_p \leftarrow CP_i(s_j)$
(3) $\quad$ **if** $s_j \in C(t)$ **then** $c_p \leftarrow c_p + 1$
(4) $\quad \tau_{p,j} \leftarrow$TARANTULA$(TP_i + 1, TF_i, c_p, CF_i(s_j))$
(5) $\quad c_f \leftarrow CF_i(s_j)$
(6) $\quad$ **if** $s_j \in C(t)$ **then** $c_f \leftarrow c_f + 1$
(7) $\quad \tau_{f,j} \leftarrow$TARANTULA$(TP_i, TF_i + 1, CP_i(s_j), c_f)$
(8) $P \leftarrow \{p_j = 0 | 1 \leq j \leq m\}$
(9) $P_{sum} \leftarrow 0$
(10) **foreach** $s_j \in S$
(11) $\quad p_j \leftarrow \frac{TF_i}{TP_i+TF_i} \cdot \frac{\tau_{f,j}}{\sum_{j=1}^{m} \tau_{f,j}} + \frac{TP_i}{TP_i+TF_i} \cdot \frac{\tau_{p,j}}{\sum_{j=1}^{m} \tau_{p,j}}$
(12) $\quad P_{sum} \leftarrow P_{sum} + p_j$
(13) $H \leftarrow \sum_{p_j \in P} -\frac{p_j}{P_{sum}} \cdot \log \frac{p_j}{P_{sum}}$
(14) **return** H

Algorithm 2: Tarantula Suspiciousness Metric
TARANTULA(tp, tf, cp, cf)
Input: Total number of passing tests, tp, total number of failing tests, tf, number of passing tests that cover the statement of focus, cp, and number of failing tests that cover the statement of focus, cf
Output: Tarantula suspiciousness metric, τ
(1) **if** $tp == 0$ **then** $r_p \leftarrow 0$
(2) $\quad$ **else** $r_p \leftarrow \frac{cp}{tp}$
(3) **if** $tf == 0$ **then** $r_f \leftarrow 0$
(4) $\quad$ **else** $r_f \leftarrow \frac{cf}{tf}$
(5) $\tau \leftarrow = \frac{r_f}{r_p+r_f}$
(6) **return** τ

4.2 FLINT Algorithm

Algorithm 3 illustrates the top-level algorithm for FLINT. The first loop executes tests in T following the prioritisation by greedy algorithm, which is widely used for coverage-based test case prioritisation [24],

Subject	# of Tests	Lines of Code	Executable Lines
`flex`	567	12,407–14,244	3,393–3,965
`grep`	199	12,653–13,363	3,078–3,314
`gzip`	214	6,576– 7,996	1,705–1,993
`sed`	360	8,082–11,990	1,923–2,172

Table 2: Subject Programs from SIR

until the first fault occurs. The second loop prioritises the set of remaining tests, R, by choosing the test with the lowest lookahead entropy value.

Algorithm 3: FLINT
FLINT(T)
(1) $index \leftarrow 0$
(2) $G \leftarrow \{\}$
(3) **while** $|G| < |T|$
(4) $t \leftarrow$ GREEDYORDER$(index)$
(5) Execute t and update TP_i, TF_i, CP_i and CF_i
(6) $G \leftarrow G \cup \{t\}$
(7) $index \leftarrow index + 1$
(8) **if** t fails **then break**
(9) $R \leftarrow T - G$
(10) **while** $|R| > 0$
(11) Pick $t \in R$ s.t. $\forall(t' \in R)(t' \neq t)(\text{EL}(t) \leq \text{EL}(t)$
(12) Execute t and update TP_i, TF_i, CP_i and CF_i
(13) $R \leftarrow R - \{t\}$

5 Experimental Setup

5.1 Subjects

Table 2 lists the subject programs studied in the paper. All four Unix utility programs are obtained from Software Infrastructure Repository (SIR) [13] along with their test suites: `flex` is a lexical analyser, `grep` is a text-search utility, `gzip` is a compression utility and `sed` is a stream text editor. We consider five consecutive versions for each program.

Since SIR only contains the fault matrices, statement coverage information was collected using the widely used GNU profiler, `gcov`. The number of executable lines in Table 2 is produced by `gcov` version 4.3.2 running on Linux version 2.6.27. Both the coverage-based test case prioritisation and FLINT have been performed using only the executable lines.

For each subject program, we selected a test suite that can be applied across all five versions. This is to ensure that, when FLINT is being applied to version n of the program, there exists matching coverage data for each test from version $n-1$. However, test suites for version 4 and 5 of `sed` were completely re-written from those for the previous versions and, therefore, could not be used for the robustness study.

5.2 Faults

SIR provides a total of 219 both real and seeded faults across the five versions of subject programs [13]. Out of 219 faults, 35 were excluded because these faults were unreachable in the compiled binary for the experimental environment. Out of the remaining 184 faults, another 92 were excluded because they were not detected by any test from the chosen test suites. The paper considers the remaining 92 faults, which are listed in Table 3.

Subject	V1	V2	V3	V4	V5
`flex`	15	14	7	9	2
`grep`	2	1	5	3	0
`gzip`	7	3	0	3	5
`sed`	0	5	6	1	4

Table 3: Number of faults studied in the paper

The robustness study uses versions from 2 to 5 of subject programs, because it requires coverage information from the previous versions. Therefore, the robustness study considers only 63 faults.

5.3 Evaluation

We compare FLINT approach and the traditional Test Case Prioritisation (TCP, hereafter) regarding their effectiveness for early fault localisation. While there is no known approach for prioritising tests for early fault localisation, we argue that TCP provides a good baseline because the aim of TCP is to maximise early fault detection. If a fault can be repeatedly detected early in testing, it should help the tester determining its locality.

In order to answer **RQ2**, we turn to a widely-studied metric that measures the effectiveness of fault localisation is *Expense* [20], which is defined as follows:

$$Expense = \frac{\text{rank of faulty statement}}{\text{number of executable statements}} \cdot 100$$

The numerator is the rank of the faulty statement when sorted according to the suspiciousness metric: the rank of tied statements are equal to the sum of the number of the tied statements and the number of statements ranked before them [20, 26]. Expense metric represents the percentage of the source code the tester has to investigate before the faulty statement is encountered. Intuitively, higher suspiciousness for the faulty statement should result in lower Expense metric. However, because of the reasons discuseed in Section 2.2, the suspiciousness metric and the Expense metric may not always agree with each other.

We execute the test suite following TCP ordering and FLINT ordering. TCP ordering is obtained using the *additional* approach with resets [14]. After executing each test for each ordering, we calculate the suspiciousness of the faulty statement and the corresponding Expense metric, thus providing a time-series of both metrics for each ordering.

The improvement in suspiciousness is observed by measuring the increment in suspiciousness achieved by FLINT over TCP ordering; the improvement in Expense is observed by measuring the reduction in Expense achieved by FLINT over TCP ordering. We report the comparison of mean values and the results of the statistical hypothesis tests. By definition, FLINT produces the same suspiciousness metric and, therefore, the same Expense metric for all statements when the entire test suite is executed.

The mean values represent the expected level of improvement when the testing is stopped at any arbitrary point. The statistical hypothesis test provides statistical confidence on the observed improvement. We categorise the results of statistical hypothesis tests into the following categories:

- **Positive with Significance(`PS`)**: the technique shows statistically significant improvement over the untreated ordering.
- **Positive with No significance(`PN`)**: the mean value of the metric does show improvement, but without statistical significance.
- **Equal(`EQ`)**: the technique performs equally well compared to the untreated ordering.

Subj.	V	Fault ID	$\bar{\tau_T}$	$\bar{\tau_F}$	T_τ	$Avg.\Delta_E$	T_E	Subj.	V	Fault ID	$\bar{\tau_T}$	$\bar{\tau_F}$	T_τ	$Avg.\Delta_E$	T_E
		F_AA_6	0.53	0.55	PS	-0.90	NS	flex	V5	F_JR_2	1.00	1.00	EQ	9.64	PS
		F_AA_1	0.78	0.76	NS	8.77	PS	grep	V1	F_KP_2	1.00	1.00	EQ	12.28	PS
		F_AA_2	1.00	1.00	EQ	-0.36	NS			F_DG_4	0.80	0.91	PS	1.92	PS
		F_AA_3	0.59	0.75	PS	-4.83	NS	grep	V2	F_DG_1	0.66	0.83	PS	3.13	PS
		F_JR_4	1.00	1.00	EQ	1.13	PS			F_KP_7	0.84	0.84	EQ	0.00	EQ
		F_JR_6	1.00	1.00	EQ	-1.13	PS			F_KP_3	0.65	0.76	PS	8.65	PS
		F_JR_5	1.00	1.00	EQ	-1.70	NS	grep	V3	F_DG_8	0.60	0.70	PS	-1.33	NS
flex	V1	F_JR_2	0.78	0.76	NS	8.77	PS			F_DG_2	0.96	0.99	PS	-1.03	NN
		F_JR_3	1.00	1.00	EQ	-5.52	NS			F_DG_3	0.75	0.85	PS	2.32	PN
		F_HD_3	1.00	1.00	EQ	-0.36	NS			F_KP_8	1.00	1.00	EQ	-1.90	NS
		F_HD_1	0.54	0.56	PS	4.20	PS	grep	V4	F_DG_3	0.62	0.62	PS	-0.00	NS
		F_HD_6	0.51	0.50	NS	9.84	PS			F_KP_6	0.98	0.90	NS	-0.68	NS
		F_HD_7	0.95	0.99	PS	-2.09	NS			F_KL_2	0.81	0.82	PS	1.12	PS
		F_HD_4	0.53	0.56	PS	0.80	PS			F_KL_6	0.41	0.42	PN	0.00	EQ
		F_HD_5	1.00	1.00	EQ	-2.14	PS			F_KP_10	0.99	0.99	EQ	0.00	EQ
		F_AA_4	0.63	0.61	NS	-2.09	NS	gzip	V1	F_KP_11	0.77	0.83	PS	0.38	PS
		F_AA_5	0.61	0.64	PS	3.70	PS			F_KP_9	1.00	1.00	EQ	-0.54	NN
		F_AA_2	0.50	0.50	EQ	4.21	PS			F_TW_3	1.00	1.00	EQ	-0.82	NS
		F_AA_3	0.99	0.99	EQ	0.00	EQ			F_KP_1	0.98	0.97	PS	-2.24	NS
		F_JR_6	0.97	0.97	NS	0.15	PS			F_KL_1	0.73	0.62	NS	0.84	PS
		F_HD_8	0.96	0.99	PS	1.38	PS	gzip	V2	F_KL_3	0.87	0.90	PS	0.90	PS
		F_JR_5	1.00	1.00	EQ	-0.02	NS			F_KL_8	0.66	0.65	NN	-0.02	NN
flex	V2	F_JR_2	0.96	0.99	PS	1.38	PS			F_KL_1	0.99	0.99	EQ	-0.00	NS
		F_JR_3	0.99	1.00	PS	-3.63	PS	gzip	V4	F_KL_8	0.96	0.94	NS	-0.34	NS
		F_JR_1	1.00	1.00	EQ	-0.02	NS			F_KP_3	0.99	0.99	EQ	0.00	EQ
		F_HD_2	1.00	1.00	EQ	-0.66	NS			F_KL_1	0.96	0.95	NS	-0.32	NS
		F_HD_6	1.00	1.00	EQ	-12.03	NS			F_KL_2	0.89	0.87	NN	-2.39	NS
		F_HD_7	1.00	1.00	EQ	-0.00	NN	gzip	V5	F_KL_4	0.91	0.91	EQ	0.00	EQ
		F_HD_4	0.69	0.71	PS	3.37	PS			F_KL_8	0.95	0.95	NS	0.00	EQ
		F_AA_4	0.50	0.50	NN	3.77	PS			F_TW_1	0.74	0.86	PS	-7.37	NS
		F_AA_5	1.00	1.00	EQ	-0.31	NS			F_AG_20	0.64	0.63	NS	0.26	PN
		F_AA_3	0.53	0.54	PS	-0.28	NS			F_AG_17	0.38	0.41	PS	0.18	PS
flex	V3	F_JR_5	1.00	1.00	EQ	-0.50	NN	sed	V2	F_AG_12	0.96	0.97	PS	1.48	PS
		F_JR_2	1.00	1.00	EQ	-0.31	NS			F_AG_19	0.92	0.85	NS	3.60	PS
		F_JR_3	1.00	1.00	EQ	-0.50	NN			F_AG_2	0.98	0.95	NS	0.18	PS
		F_HD_6	1.00	1.00	EQ	-0.31	NS			F_AG_15	0.94	0.95	PS	0.29	PS
		F_AA_7	0.98	0.99	PS	1.56	PS			F_AG_5	0.88	0.91	PS	0.00	EQ
		F_AA_1	0.81	0.89	PS	0.81	PS	sed	V3	F_AG_17	0.99	0.98	NN	-0.60	NS
		F_AA_2	1.00	1.00	EQ	-1.41	NS			F_AG_6	1.00	1.00	EQ	0.26	NS
		F_AA_3	0.54	0.63	PS	8.31	PS			F_AG_11	0.98	0.99	PS	0.20	PS
flex	V4	F_JR_4	0.99	0.99	PS	0.09	PS			F_AG_18	0.97	0.97	EQ	0.03	PS
		F_JR_2	1.00	1.00	EQ	-2.06	NS	sed	V4	F_KRM_2	0.94	0.94	EQ	0.00	EQ
		F_JR_3	1.00	1.00	EQ	1.12	PS			F_KRM_8	0.99	0.99	NS	0.73	PS
		F_JR_1	0.54	0.63	PS	8.31	PS	sed	V5	F_KRM_1	0.94	0.94	NS	-0.87	PS
		F_HD_5	1.00	1.00	EQ	1.12	PS			F_KRM_2	1.00	1.00	EQ	-0.21	PS
flex	V5	F_AA_4	0.89	0.91	PS	4.18	PS			F_KRM_10	0.87	0.89	PS	0.00	EQ

Table 4: Statistical Analysis for Precision Study: **PS** and **PN** denote that FLINT achieves improvements over TCP with and without statistical significance, respectively. Similarly, **NS** and **NN** denote degeneration with and without statistical significance. **EQ** denotes that TCP and FLINT produces the same results. Excluding 37 faults that do not provide any room for improvement of suspiciousness (because $\tau = 1.0$ regardless of test ordering for these faults), FLINT achieves higher suspiciousness with statistical significance for 64% of faults (35 out of 55). However, this does not always translate into lower Expense, as can be seen from the results of the statistical hypothesis testing for Expense (T_E) due to inherent limitations in coverage-based suspiciousness metrics described in Section 2.2 and Section 5.3.

- **Negative with No significance(NN)**: the mean value of the metric does show degeneration, but without statistical significance.

- **Negative with Significance(NS)**: the technique shows statistically significant degeneration from the untreated ordering.

Category **EQ** is possible when, for example, the faulty statement is detected by the first test and its suspiciousness remains 1.0 throughout the testing, regardless of the ordering of tests: any ordering produced by TCP or FLINT will always result in the same suspiciousness values.

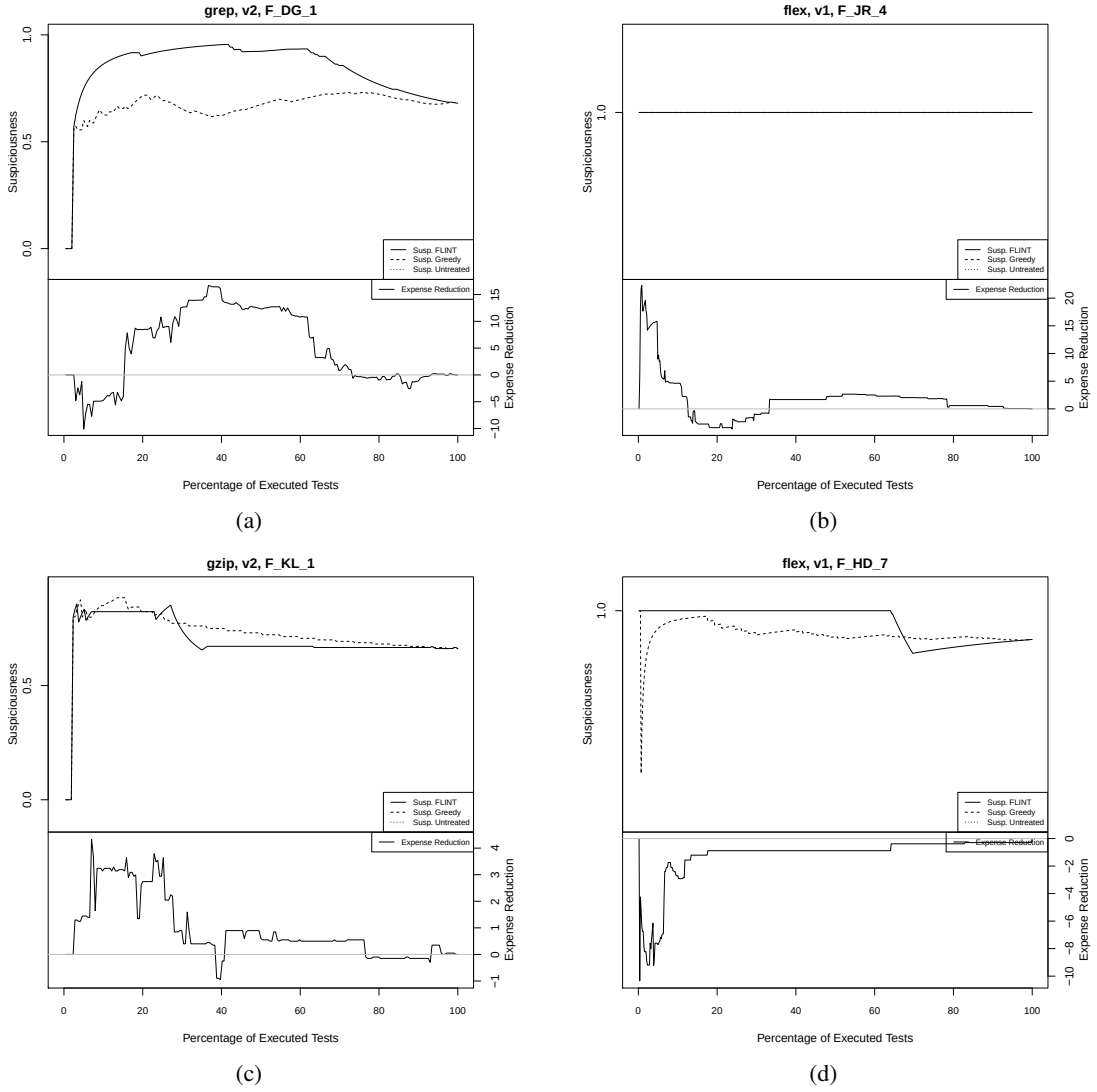

Figure 1: Plots of suspiciousness and Expense reduction from the precision study. Figure 1(a) represent the cases when FLINT approach produces higher suspiciousness and lower Expense, compared to TCP approach, with statistical significance. Figure 1(b) and 1(c) represent the cases when the reduction in entropy achieved by FLINT results in reduced Expense, despite the fact that the suspiciousness value of the faulty statement is equal to that from TCP approach (Figure 1(b)) or even lower (Figure 1(c)). This is because the reduction in entropy is achieved by *lowering* the suspiciousness metric for non-faulty statements. Howevver, in Figure 1(d), the suspiciousness metric is not in alignment with Expense: the increased suspiciousness metric resulted in higher Expense.

6 Results

6.1 Precision Study

Table 4 contains the results from the precision study as well as the statistical analysis. Column $\bar{\tau}_T$ and $\bar{\tau}_F$ contain the mean suspiciousness of the faulty statement, over the entire test suite, by TCP and FLINT respectively. Column $Avg.\Delta_{E_T}$ contains the mean reduction in Expense metric achieved by FLINT over TCP.

For FLINT to produce more effective fault localisation, it should provide the tester with higher suspiciousness and lower Expense metric for the faulty statement. This is analysed using Mann-Whitney 'U' test. The

Mann-Whitney 'U' test is a non-parametric statistical hypothesis test, i.e. it allows the comparison of two samples with unknown distributions. Column T_τ contains the result classification of the Mann-Whitney 'U' test for the suspiciousness metric. The null hypothesis is that there is no difference between $\bar{\tau}_F$ and $\bar{\tau}_T$; the alternative hypothesis is that $\bar{\tau}_F > \bar{\tau}_T$. Similarly, column T_E contains the result classification of the Mann-Whitney 'U' test for the Expense metric. The null hypothesis is that there is no difference between the mean Expense from FLINT and the mean Expense from TCP. The alternative hypothesis is that the mean Expense from FLINT is lower than the mean Expense from TCP. The confidence level is 95%.

Table 5 contains the classification of the statistical hypothesis testing for the precision study. Out of 92 studied faults, 37 were classified into `EQ` category for T_τ. For these faults, the suspiciousness of the faulty statement remains at 1.0 from the point of fault detection to the end of the test suite execution, regardless of test ordering. That is, it is not possible to make improvement in suspiciousness metric. Excluding these 37 faults for which there is no room for improvement of suspiciousness metric, FLINT achieves statistically significant increment of suspiciousness for 64% of the studied faults.

However, the number of faults classified into `EQ` category for Expense metric is smaller than those for suspiciousness metric. This is because the Expense of the faulty statement depends not only on the suspiciousness of the statement itself but also on the suspiciousness of other statements. It should be also noted that the 35 faults that were classified into `PS` category for Expense are not the same set of faults that are in `PS` category for suspiciousness. For example, Table 4 shows that the fault `F_AA_6` for the version 1 of `flex` is `PS` for suspiciousness but `NS` for Expense.

Figure 1 provides more detailed explanation with exemplar cases. The top pane shows how the suspiciousness metric for the faulty statement has changed during the execution of test orderings from FLINT and TCP approach. The bottom pane shows the reduction in Expense ($E_T - E_F$).

Figure 1(a) shows a case when both suspiciousness and Expense result in `PS` category. FLINT produces higher suspiciousness during the most of the duration of the testing, which results in reductions in Expense for the most of times.

Figure 1(b) and 1(c) represent two interesting cases when reduction in Expense is achieved despite the fact that the suspiciousness metric for the faulty statement from FLINT is the same, at 1.0, as that from TCP approach (Figure 1(b)) or even lower with statistical significance (Figure 1(c)). These results area achieved because choosing a test that produces the lowest entropy may not only increase the suspiciousness of the faulty statement but also lower the suspiciousness of the non-faulty statements (Section 3.1.2).

However, the results also show that increment of the suspiciousness metric may not always result in reduction in Expense, as discussed in Section 2.2 and Section 5.3. In Figure 1(d), although τ_F of fault `F_HD_7` for version 1 of `flex` is higher with statistical significance than τ_T, FLINT fails to make reductions in Expense. This means that, when considering the impact of FLINT on Expense reduction, we cannot exclude faults in `EQ` category.

Metric	PS	PN	EQ	NN	NS
Suspiciousness	35	1	37	4	15
Expense	43	2	10	6	31

Table 5: Hypothesis Test for Precision Study

To answer **RQ1**, Table 4 and 5 provide evidence that FLINT achieves higher suspiciousness with statistical significance for 64% of studied faults for which there exist room for improvement. It produces lower suspiciousness with statistical significance for only 27% of the faults.

To answer **RQ2**, FLINT reduces Expense metric with statistical significance for 43 faults out of 92 studied (47%). However, these 35 faults are not the same faults for which FLINT increased the suspiciousness metric, as can be seen in Figure 1(c). For these faults, the average reduction over the entire test suite ranges

Subj.	V	Fault ID	$\bar{\tau_T}$	$\bar{\tau_F}$	T_τ	$Avg.\Delta_E$	T_E	Subj.	V	Fault ID	$\bar{\tau_T}$	$\bar{\tau_F}$	T_τ	$Avg.\Delta_E$	T_E
flex	V2	F_AA_4	0.63	0.61	NS	1.69	PS	grep	V2	F_DG_1	0.66	0.83	PS	8.37	PS
		F_AA_5	0.60	0.63	PS	2.46	PS	grep	V3	F_KP_7	0.84	0.84	EQ	0.00	EQ
		F_AA_2	0.50	0.50	EQ	4.29	PS			F_KP_3	0.65	0.76	PS	8.80	PS
		F_AA_3	0.96	0.96	EQ	0.00	EQ			F_DG_8	0.59	0.70	PS	0.26	PS
		F_JR_6	0.97	0.98	NS	0.17	PS			F_DG_2	0.96	0.99	PS	-1.45	NN
		F_HD_8	0.95	0.96	PS	4.02	PS			F_DG_3	0.75	0.86	PS	3.53	PS
		F_JR_5	1.00	1.00	EQ	0.02	NS	grep	V4	F_KP_8	1.00	1.00	EQ	-1.74	NS
		F_JR_2	0.95	0.96	PS	4.02	PS			F_DG_3	0.82	0.81	NS	-0.12	NS
		F_JR_3	0.99	0.99	NS	0.42	PS			F_KP_6	0.98	0.90	NS	-0.70	NS
		F_JR_1	1.00	1.00	EQ	0.02	NS	gzip	V2	F_KL_1	0.73	0.59	NS	0.34	NS
		F_HD_2	1.00	1.00	EQ	-1.17	NS			F_KL_3	0.91	0.95	PS	0.63	PS
		F_HD_6	1.00	1.00	EQ	-2.92	NS			F_KL_8	0.66	0.55	NS	0.31	PN
		F_HD_7	1.00	1.00	EQ	0.02	PN	gzip	V4	F_KL_1	0.99	0.99	EQ	0.00	EQ
		F_HD_4	0.69	0.71	NS	3.45	PS			F_KL_8	0.96	0.95	NS	-0.32	NS
flex	V3	F_AA_4	0.50	0.50	NN	3.76	PS			F_KP_3	0.99	0.99	EQ	0.00	EQ
		F_AA_5	1.00	1.00	EQ	-0.31	NS	gzip	V5	F_KL_1	0.96	0.94	NS	-0.33	NS
		F_AA_3	0.53	0.54	PS	-0.28	NS			F_KL_2	0.89	0.86	NN	-2.98	NS
		F_JR_5	1.00	1.00	EQ	-0.48	NN			F_KL_4	0.93	0.93	EQ	0.00	EQ
		F_JR_2	1.00	1.00	EQ	-0.31	NS			F_KL_8	0.96	0.95	NS	0.00	EQ
		F_JR_3	1.00	1.00	EQ	-0.48	NN			F_TW_1	0.72	0.63	NS	-0.40	PS
		F_HD_6	1.00	1.00	EQ	-0.31	NS	sed	V2	F_AG_20	0.65	0.60	NS	-0.45	NS
flex	V4	F_AA_7	0.98	0.99	PS	1.57	PS			F_AG_17	0.38	0.40	PS	0.17	PS
		F_AA_1	0.81	0.87	PS	0.70	PS			F_AG_12	0.95	0.98	PS	-0.64	NS
		F_AA_2	1.00	1.00	EQ	-1.41	NS			F_AG_19	0.92	0.94	PS	2.07	PN
		F_AA_3	0.54	0.63	PS	8.26	PS			F_AG_2	0.98	0.94	NS	0.08	PS
		F_JR_4	0.99	1.00	PS	-3.68	PS	sed	V3	F_AG_15	0.94	0.95	PS	-0.34	NS
		F_JR_2	1.00	1.00	EQ	-5.97	NS			F_AG_5	0.87	0.91	PS	0.01	PS
		F_JR_3	1.00	1.00	EQ	-0.73	NN			F_AG_17	0.99	0.99	NS	-2.08	NS
		F_JR_1	0.54	0.63	PS	8.26	PS			F_AG_6	1.00	1.00	EQ	0.54	PS
		F_HD_5	1.00	1.00	EQ	-0.73	NN			F_AG_11	0.97	0.98	PS	4.00	PS
flex	V5	F_AA_4	0.89	0.90	NS	4.24	PS			F_AG_18	0.98	0.98	EQ	0.03	PS
		F_JR_2	1.00	1.00	EQ	9.64	PS								

Table 6: Statistical Analysis for Robustness Study: **PS** and **PN** denote that FLINT achieves improvements over TCP with and without statistical significance, respectively. Similarly, **NS** and **NN** denote degeneration with and without statistical significance. **EQ** denotes that TCP and FLINT produces the same results. Excluding 24 faults that do not provide any room for improvement of suspiciousness (because $\tau = 1.0$ regardless of test ordering for these faults), FLINT achieves higher suspiciousness with statistical significance for 54% of faults (21 out of 39). However, as observed in the precision study, this does not always translate into lower Expense.

from 0.15% to 12.28%. FLINT increases Expense metric with statistical significance for 31 faults out of 92 studied (34%).

6.2 Robustness Study

Table 6 contains the results from the robustness study as well as the statistical analysis. The results of the statistical hypothesis tests are summarised in Table 7. Excluding the faults in EQ, FLINT achieves higher suspiciousness for 54% of the faults studied. It also achieves statistically significant Expense reduction for 44% of the faults studied.

Figure 2 presents the representative outcome of the robustness study. Figure 2(a) and 2(c) contain the plots for the same faults depicted in Figure 1(a) and 1(c) We can observe not identical, but similar patterns in the suspiciousness plot: it shows that FLINT can cope with the coverage information from the previous version for these faults. The results shown in these two plots, as well as in Figure 2(b), are positive.

Figure 2(d) shows that there are certain stopping points for which FLINT can reduce the Expense metric even it fails to make statistically significant reduction over the entire test suite. If the testing were to be terminated within the first 10% of the test suite, FLINT would reduction Expense.

To answer **RQ3**, Table 6 and 7 shows that FLINT achieves higher suspiciousness and reduced Expense with statistical significance for more of the studied faults than TCP approach even when the coverage information from the previous version is used. Comparing the results of the robustness study with those of

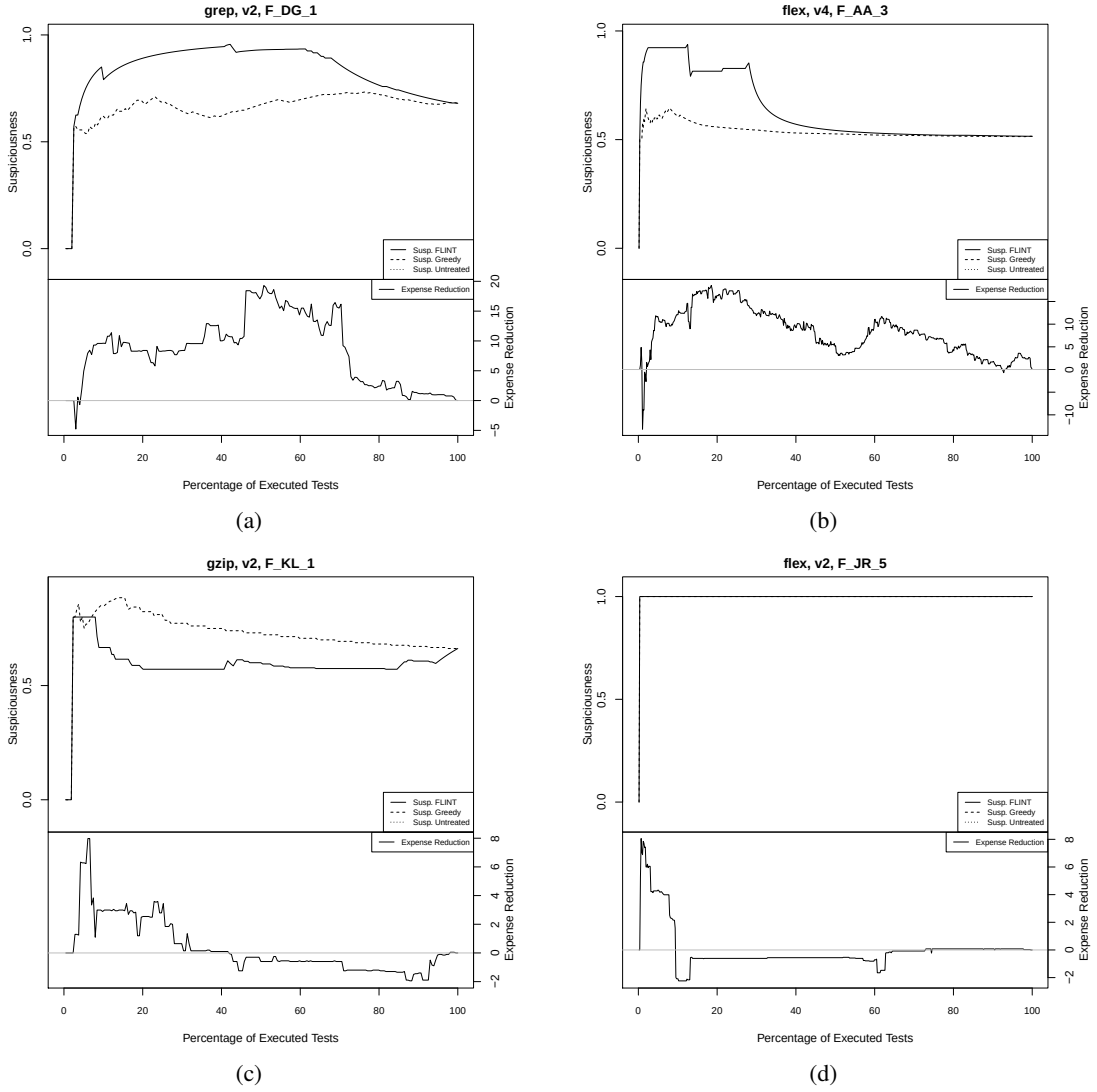

Figure 2: Plots of suspiciousness and Expense reduction from the robustness study. Figure 2(a) and 2(c) correspond to Figure 1(a) and 1(c) respectively: FLINT achieves improved suspiciousness and reduced Expense for these faults, despite the use of coverage information from the previous version for the lookahead. Figure 2(b) shows another positive result. Figure 2(d) shows a case when reduction in Expense is possible in early stage of the testing, even though the mean E_F is lower than the mean E_T with statistical significance.

the precision study provides evidence that it is possible to use FLINT with coverage information from the previous version and still achieve improvements in both suspiciousness and Expense metrics.

To answer **RQ4**, we compare the mean reductions of Expense from two consecutive versions of the same program. The third column in Table 8 shows the mean reductions of Expense for all faults in version n from Table 4. The fifth column shows the mean reductions of Expense for all faults in the corresponding version $n+1$ from Table 6. If there exists a positive correlation between the two sets of values, it becomes possible for the tester to try FLINT with known data from the previous version (following the precision study) to see whether FLINT can achieve reductions for the current version (following the robustness study). The result of Spearman's rank correlation analysis is $\rho = 0.6190$ with $p = 0.057$, supporting a positive answer to **RQ4**.

Metric	PS	PN	EQ	NN	NS
Suspiciousness	21	0	24	2	16
Expense	28	3	6	5	21

Table 7: Hypothesis Test for Robustness Study

Subject	Ver. n	$Avg.\Delta_E$	Ver. $n+1$	$Avg.\Delta_E$
flex	3	0.22	4	0.70
flex	4	1.98	5	6.94
grep	1	7.10	2	8.37
grep	2	3.13	3	2.23
grep	3	1.72	4	-0.85
gzip	1	-0.30	2	0.43
gzip	4	-0.13	5	-0.74
sed	2	1.14	3	0.36

Table 8: Comparison of Mean Expense Reduction between consecutive versions. Spearman's rank correlation coefficient ρ is 0.6190 with $p = 0.057$.

7 Related Work

Test case prioritisation is a regression testing technique that aims to maximise the rate of fault detection if the testing is terminated at an arbitrary point [24]. Since the fault information is not known, test case prioritisation techniques often rely on surrogates, for which structural coverage is widely used [14, 21]. However, test case prioritisation techniques do not separately consider the case when a fault is actually detected during the execution of the prioritised test suite.

Fault localisation is a debugging technique that aims to aid the tester to locate a detected fault [20]. Existing work focus on coverage-based metrics, program spectra analysis or Program Dependence Graph (PDG) to locate faults [1, 10, 15, 19, 20]. There is existing work that investigates the impact of test suite reduction on fault localisation [26] the impact of test case prioritisation on fault localisation has not been studied before. Recent work uses a probabilistic causal inference model for better fault localisation [4] but this paper is the first to introduce Information Theory to fault localisation.

Information theory [11], now an extensive branch of probability theory with many applications, was famously founded by Claude Shannon in a single paper [22]. It has been applied in many research areas related to computer science including machine learning, analysis of algorithms, and data mining. Applications to software engineering and particularly to programming languages, have been less common. Software metrics [2] and software evolution [3] are both areas which have seen contributions but the most active area at the present time is program analysis for quantifying information flow.

Questions about quantified information flow (QIF) arise naturally in the theory of dependence, particularly in the theory of security, in order to measure the strength of dependence (e.g. for potential covert channels). It is not surprising that one of the the earliest applications of Shannon information to programming languages was in Denning's 1982 book on cryptography and data security [12] where it appears in an informal discussion of how to analyse program constructs in terms of information flow, along with an attempt to define flow quantity. Although it subsequently became fashionable to use information theory in discussions of security properties for software systems, the first automatable analysis for QIF did not appear until 2002 [7]. This latter work was extended to a Turing complete language [8] and to a process language [5]. In the last five years a vibrant community of researchers into QIF has developed but to date all the applications have been to flow security.

Our paper introduces a novel application of Shannon entropy to the analysis of programs and has potential for extension to a theoretical framework for a probabilistic approach to testing. It is not QIF based but it adopts a similar approach to Clarkson et alia's Bayesian-influenced paper on quantifying information flow [9]. The paper describes a security attack in which a series of experiments successively update the attacker's belief about the probability distribution on the space of secrets with the aim of refining that

probability distribution to the one in which the actual secret input to the program has probability 1 while all others have probability 0.

8 Conclusion

This paper presents the first use of Information Theory for fault localisation. We build an entropy model for the locality of fault: the probability distribution of the locality of the fault is approximated using existing fault localisation metrics. The proposed technique, FLINT, aims to improve the effectiveness of fault localisation by trying to reduce the Shannon entropy of the locality of the fault. While the paper considers the use of Tarantula metric, any fault localisation metric can be plugged into FLINT.

We evaluate FLINT by evaluating its effectiveness for a novel problem: test case prioritisation for early fault localisation. Once a test suite, prioritised for early fault detection, indeed detects a fault, we switch to FLINT approach to maximise the chance of early fault localisation even if testing is terminated prematurely. Empirical evaluation of FLINT shows that it is possible to increase the suspiciousness metric and reduce the fault localisation cost for more than half of the studied faults.

References

[1] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. On the accuracy of spectrum-based fault localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, pages 89–98. IEEE Computer Society, 2007.

[2] E. B. Allen and T. M. Khoshgoftaar. Measuring coupling and cohesion: An information-theory approach. In *IEEE METRICS*, 1999.

[3] T. Arbuckle. Studying software evolution using artefacts' shared information content. *Science of Computer Programming*, In Press:–, 2010.

[4] G. K. Baah, A. Podgurski, and M. J. Harrold. Causal inference for statistical fault localization. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA 2010)*, pages 73–84. ACM Press, July 2010.

[5] M. Boreale, D. Clark, and D. Gorla. A semiring-based trace semantics for processes with applications to information leakage analysis. In *Proceedings of the 6th IFIP TC 1/WG 2.2 International Conference TCS 2010, Part of WCC2010 Proceedings*. Springer, 2010.

[6] F. P. Brooks, Jr. *The Mythical Man Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, Reading , MA , USA, 1975.

[7] D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science*, 59, 2002.

[8] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321 – 372, 2007.

[9] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.

[10] H. Cleve and A. Zeller. Locating causes of program failures. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 342–351, New York, NY, USA, 2005. ACM.

[11] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Interscience, 1991.

[12] D. E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[13] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[14] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2000)*, pages 102–112. ACM Press, August 2000.

[15] J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th International Conference on Automated Software Engineering (ASE2005)*, pages 273–282. ACM Press, 2005.

[16] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*, pages 467–477, New York, NY, USA, 2002. ACM.

[17] J.-M. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the 24th International Conference on Software Engineering*, pages 119–129. ACM Press, May 2002.

[18] Z. Li, M. Harman, and R. M. Hierons. Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering*, 33(4):225–237, 2007.

[19] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 15–26, New York, NY, USA, 2005. ACM.

[20] M. Renieres and S. Reiss. Fault localization with nearest neighbor queries. In *Proceedings of the 18th International Conference on Automated Software Engineering*, pages 30 – 39, October 2003.

[21] G. Rothermel, R. J. Untch, and C. Chu. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, October 2001.

[22] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[23] P. Tonella, P. Avesani, and A. Susi. Using the case-based ranking methodology for test case prioritization. In *Proceedings of the 22nd International Conference on Software Maintenance (ICSM 2006)*, pages 123–133. IEEE Computer Society, July 2006.

[24] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. *Software Testing, Verification, and Reliability*, *to appear*, 2010.

[25] S. Yoo, M. Harman, P. Tonella, and A. Susi. Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2009)*, pages 201–211. ACM Press, July 2009.

[26] Y. Yu, J. A. Jones, and M. J. Harrold. An empirical study of the effects of test-suite reduction on fault localization. In *Proceedings of the International Conference on Software Engineering (ICSE 2008)*, pages 201–210. ACM Press, May 2008.