Clustering Test Cases to Achieve Effective & Scalable Prioritisation Incorporating Expert Knowledge

Shin Yoo & Mark Harman
King's College London, Centre for Research on Evolution, Search & Testing (CREST)
Strand, London
WC2R 2LS, UK

Paolo Tonella & Angelo Susi
FBK-IRST
Via Sommarive, 18
38050 Povo, Trento, ITALY

ABSTRACT

Pair-wise comparison has been successfully utilised in order to prioritise test cases by exploiting the rich, valuable and unique knowledge of the tester. However, the prohibitively large cost of the pair-wise comparison method prevents it from being applied to large test suites. In this paper, we introduce a cluster-based test case prioritisation technique. By clustering test cases, based on their dynamic runtime behaviour, we can reduce the required number of pair-wise comparisons significantly. The approach is evaluated on seven test suites ranging in size from 154 to 1,061 test cases. We present an empirical study that shows that the resulting prioritisation is more effective than existing coverage-based prioritisation techniques in terms of rate of fault detection. Perhaps surprisingly, the paper also demonstrates that clustering (even without human input) can outperform unclustered coverage-based technologies, and discusses an automated process that can be used to determine whether the application of the proposed approach would yield improvement.

Categories and Subject Descriptors: D.2.5 [Software Engineering]: Testing and Debugging

General Terms: Algorithms

Keywords: Test case prioritisation, AHP, Clustering

1. INTRODUCTION

Test case prioritisation seeks to find an efficient ordering of test case execution for regression testing. The most ideal ordering of test case execution is one that reveals faults earliest. Since the nature and location of actual faults are generally not known in advance, test case prioritisation techniques have to rely on available surrogates for prioritisation criteria. Structural coverage, requirement priority and mutation score have all previously been utilised as criteria for test case prioritisation [3, 6, 18]. However, there is no single prioritisation criterion whose results dominate the others.

One potentially powerful way to enhance a prioritisation criterion is to utilise domain expert judgement by asking the human tester to compare the importance of different test cases. A competent human tester can provide rich domain knowledge about the System Under Test (SUT), including knowledge about logical effects of recent changes and rationale behind the existing test cases. Human guidance may also be required in order to take account of the many implicit, unstated reasons that the tester may have for favouring one test case over another. If this human guidance is not accounted for, the tester may reject the proposed order suggested by a prioritisation algorithm.

Prioritisation involving human judgement is not new. The Operations Research community has developed techniques including the Analytic Hierarchy Process (AHP) algorithm [17] that help decision makers to prioritise tasks. However, prioritisation techniques that involve humans present scalability challenges. A human tester can provide consistent and meaningful answers to only a limited number of questions, before fatigue starts to degrade performance. Previous empirical studies show that the largest number of pair-wise comparisons a human can make consistently is approximately 100 [1]. Unfortunately, useful test suites often contain many test cases, potentially requiring considerably more than 100 comparisons.

To address this problem, this paper uses clustering algorithms to reduce the cost of human-interactive prioritisation. In our approach, the human tester prioritises, not the individual test cases, but clusters of 'similar' test cases. With a very simple clustering technique, such as agglomerative hierarchical clustering, it is possible to generate an arbitrary number of clusters. This allows for control of the number of comparisons presented to the human tester. The reduced number of required comparisons makes it feasible to apply expert-guided prioritisation techniques to much larger data sets. The paper presents results on the scalability potential of this clustering approach.

The AHP-based prioritisation technique is empirically compared to coverage-based prioritisation techniques using the APFD (Average Percentage of Fault Detection) metric. In order to model various possible human behaviours, we introduce an error model. This allows us to empirically explore the robustness of our approach in the presence of varying degrees of human 'bias' (giving guidance that draws the algorithm away from fault finding test cases). The results show that AHP-based prioritisation is robust; it can outperform coverage-based prioritisation even when the human tester provides misleading answers to comparison questions.

The primary contributions of this paper are as follows:

1. The paper presents a novel use of clustering in test case prioritisation. The use of clustering enables us to apply the interactive prioritisation technique, AHP.
2. The paper presents a novel prioritisation technique which is based on AHP. Although AHP has been widely adopted by the Requirements Engineering community, its application

ISSTA'09, July 19–23, 2009, Chicago, Illinois, USA.

in test case management is new. The results of the empirical study show that AHP-based prioritisation can outperform coverage-based prioritisation.

3. The paper presents a more realistic model of human behaviour by introducing an error model. Using the error model, we analyse the threshold consistency level required for human testers. The results show that our approach is robust. As the paper demonstrates, the technique can also provide guidelines on whether AHP-based prioritisation is appropriate for a testing project or not.

4. The paper presents an automated process that can determine whether the cluster-based prioritisation approach will be effective against a specific pair of SUT and test suite. Using this automated process, the human tester can decide whether committing human effort would be worthwhile for a particular testing task.

The rest of the paper is organised as follows. Section 2 introduces the cluster-based prioritisation technique used in the paper. Section 3 describes the Analytic Hierarchy Process and analyses the impact on cost that clustering can bring about. Section 4 explains the details of the empirical study, the results of which are presented in Section 5. Section 6 presents related work and Section 7 concludes.

2. CLUSTERING BASED PRIORITISATION

2.1 Motivation

A pair-wise comparison approach for prioritisation requires $O(n^2)$ comparisons. While redundancy may make pair-wise comparison very robust, the high cost has prevented it from being applied to test case prioritisation. For example, AHP has been well studied in the Requirements Engineering field. The maximum number of comparisons a human can make consistently is approximately 100 [1]; above this threshold, inconsistency grows significantly, leading to reduced effectiveness.

In order to require less than 100 pair-wise comparisons, the test suite could contain no more than 14 test cases. Considering the scale of real world testing projects, the scalability issue presents a significant challenge. For example, suppose there are 1,000 test cases to prioritise; the total number of required pair-wise comparisons would be 499,500. It is clearly unrealistic to expect a human tester to provide reliable responses for such a large number of comparisons.

This paper aims to reduce the number of comparisons required for the pair-wise comparison approach through the use of clustering. Instead of prioritising individual test cases, clusters of test cases are prioritised using techniques such as AHP. From the prioritised clusters, the ordering between individual test cases is then generated.

2.2 Clustering Criterion

The clustering process partitions objects into different subsets so that objects in each group share common properties. The clustering criterion determines which properties are used to measure the commonality. When considering test case prioritisation, the ideal clustering criterion would be the similarity between the faults detected by each test case. However, this information is inherently unavailable before the testing task is finished. Therefore, it is necessary to find a surrogate for this, in the same way as existing coverage-based prioritisation techniques turn to surrogates for fault-detection capabilities.

In this paper we utilise dynamic execution traces of each test case as a surrogate for the similarity between features tested. Execution of each test case is represented by a binary string. Each bit corresponds to a statement in the source code. If the statement has been executed by the test case, the digit is 1; otherwise it is 0. The similarity between two test cases is measured by the distance between two binary strings using Hamming distance.

2.3 Clustering Method

We use a simple agglomerative hierarchical clustering technique. Its pseudo-code is described in Algorithm 1 below:

Algorithm 1: Agglomerative Hierarchical Clustering
Input: A set of n test cases, T
Output: A dendrogram, D, representing the clusters
(1) Form n clusters, each with one test case
(2) $C \leftarrow \{\}$
(3) Add clusters to C
(4) Insert n clusters as leaf node into D
(5) **while** there is more than one cluster
(6) Find a pair of clusters with minimum distance
(7) Merge the pair into a new cluster, c_{new}
(8) Remove the pair of test cases from C
(9) Add c_{new} to C
(10) Insert c_{new} as a parent node of the pair into D
(11) **return** D

The resulting dendrogram is a tree structure that represents the arrangement of clusters. Figure 1 shows an example dendrogram. It is possible to generate k clusters for any k in $[1, n]$ by cutting the tree at different heights.

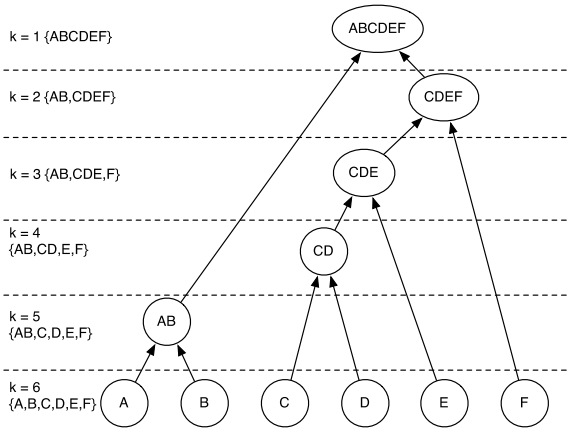

Figure 1: An example dendrogram from agglomerative hierarchical clustering. Cutting the tree at different height produces different number of clusters.

2.4 Interleaved Clusters Prioritisation

Prioritisation of a clustered test suite is a different problem from the traditional test case prioritisation problem. Two separate layers of prioritisation are required in order to prioritise a clustered test suite. *Intra-cluster* prioritisation is prioritisation of test cases that belong to the same cluster, whereas *inter-cluster* prioritisation is prioritisation of clusters. This paper introduces the Interleaved Clusters Prioritisation (ICP) process that uses both layers of prioritisation.

It would be more advantageous to *interleave* clusters of test cases than to execute an entire cluster before executing the next. The latter approach would result in repeatedly executing similar parts of

SUT before the prioritisation technique chooses the next cluster; if these test cases reveal similar sets of faults, the rate of fault detection would be less than ideal because the prioritisation technique will reveal a similar set of faults repeatedly. The former approach will avoid this by switching clusters whenever it chooses the next test case.

In ICP, intra-cluster prioritisation is performed first. Based on the results of intra-cluster prioritisation, each cluster is assigned a test case that represents the cluster. Using these representatives, ICP performs inter-cluster prioritisation. The final step is to interleave prioritised clusters using the results of both intra- and inter-cluster prioritisation.

More formally, suppose a test suite TS is clustered into k clusters, $C_1, \ldots, C_k$. After intra-cluster prioritisation, we obtain ordered sets of test cases, $OC_1, \ldots, OC_k$. Let $OC_i(j)$ be the jth test case in cluster OC_i. Each ordered set OC_i is then represented by $OC_i(1)$ in the inter-cluster prioritisation, which will produce OOC, an ordered set of $OC_i (1 \leq i \leq k)$. Let OOC_i be the ith cluster in OOC. The interleaving process is described in pseudo-code in Algorithm 2.

Algorithm 2: Interleaved Clusters Prioritisation
Input: An ordered set of k ordered clusters, OOC
Output: An ordered set of test cases, OTC
(1) OTC =<>
(2) $i \leftarrow 1$
(3) **while** OOC is not empty
(4) Append $OOC_i(1)$ to OTC
(5) Remove $OOC_i(1)$ from OOC_i
(6) **if** OOC_i is empty **then** Remove OOC_i from OOC
(7) $i \leftarrow (i + 1) \bmod k$
(8) **return** OOC

For example, given $OOC =<< t_3, t_1 >, < t_4, t_2 >, < t_5 >>$, the result of Algorithm 2 will be a sequence of test cases, $< t_3, t_4, t_5, t_1, t_2 >$. Note that ICP does not presume any specific choice of prioritisation technique. Any existing test case prioritisation technique can be used for either intra-cluster or inter-cluster prioritisation.

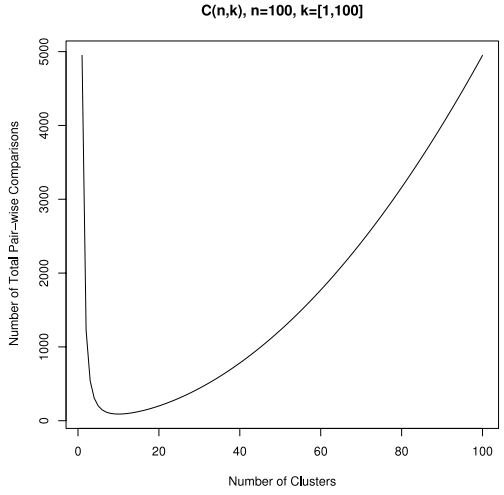

Figure 2: Plot of average number of pair-wise comparisons required for k cluster-based prioritisation of 100 test cases.

2.5 Cost of Pair-wise Comparisons

Since pair-wise comparisons require human intervention, the cost of any pair-wise comparison approach largely depends on the number of comparisons required. When the pair-wise comparison approach is used both for intra- and inter-cluster prioritisation, the number of comparisons required for ICP is the sum of the cost of intra-cluster prioritisation and inter-cluster prioritisation. Given a test suite of size n clustered into k clusters, each cluster contains $\frac{n}{k}$ test cases on average. The average number of comparisons for intra-cluster prioritisation is $k \cdot \frac{1}{2}\frac{n}{k}(\frac{n}{k} - 1)$. The number of comparisons of inter-cluster prioritisation is computed simply as $\frac{k(k-1)}{2}$. Therefore, the average total cost of pair-wise ICP for a test suite of size n and k clusters, $C(n, k)$, is $\frac{k(k-1)}{2} + k \cdot \frac{\frac{n}{k}(\frac{n}{k}-1)}{2}$.

For all positive n, there exists a specific value of k that minimises $C(n, k)$. Figure 2 illustrates $C(n, k)$ when $n = 100$ and $1 \leq k \leq n$. The maximum cost, with no clustering, is 4,950 comparisons. With clustering, the minimum cost is 381 when $k = 17$. While the reduction is by an order of magnitude, the minimum cost of 381 is still too expensive for a human tester to consider.

To further reduce the cost, ICP used in the paper is hybridised so that intra-cluster prioritisation uses the traditional coverage-based greedy prioritisation algorithm. The human tester is only involved with inter-cluster prioritisation. The cost of hybrid approach is, therefore, only the number of comparisons required for inter-cluster prioritisation, which is $C(n, k) = \frac{k(k-1)}{2}$. To ensure that fewer than 100 comparisons are required, we use hybrid-ICP with $k = 14$ throughout the paper, which results in 91 comparisons.

2.6 Suitability Test

ICP is most effective when the result of clustering is semantically significant, i.e. test cases that execute similar parts of SUT belong to the same cluster. As k decreases, the semantic significance of clustering is also diminished, since eventually the clustering algorithm will start to place semantically different test cases in the same cluster. Therefore, hybrid ICP with $k = 14$ may not work well with every test suite/SUT combination.

Since any form of human involvement in test case prioritisation is a significant commitment, applying hybrid ICP to a combination of test suite and SUT that is not suitable would be a waste of resources. A decision is required as to whether it is worth applying the hybrid ICP. To support this decision making process, we propose an automated suitability test that does not require human judgement. The test is an automated ICP, fully based on structural coverage. Both intra- and inter-cluster prioritisation is performed based on structural coverage. It also uses fault detection information using faults that belong to the AR (Already Revealed) fault set. If the result of the test is not worse than traditional coverage-based prioritisation techniques, it would confirm that clustering is not detrimental to the performance of ICP, in which case replacing inter-cluster prioritisation with the pair-wise comparison approach is likely to have a positive impact on the rate of fault detection with the unknown faults that belong to the TBR (To Be Revealed) fault set.

3. ANALYTIC HIERARCHY PROCESS

3.1 Definition

In order to prioritise n items, AHP requires all possible pair-wise comparisons between n items. Comparisons are represented using the scale of preference described in Table 1.

Note that the preference relation is not necessarily transitive. The decision maker is entitled to give answers such as $A \succ B, B \succ C$ and $C \succ A$. In other words, p_{ik} is not necessarily equal to $p_{ij} \cdot p_{jk}$. This lack of transitivity in the preference relation allows AHP to cope with inconsistencies given by the decision maker. However, these inconsistencies are mitigated by the high redundancy available due to multiple comparisons. By definition, the scale is a ratio-based measurement. That is, given p_{ij}, p_{ji} is defined as $\frac{1}{p_{ij}}$.

p_{ij}	Preference
1	i is equally preferable to j
3	i is slightly preferable over j
5	i is strongly preferable over j
7	i is very strongly preferable over j
9	i is extremely preferable over j

Table 1: Scale of preference used in the comparison matrix of AHP

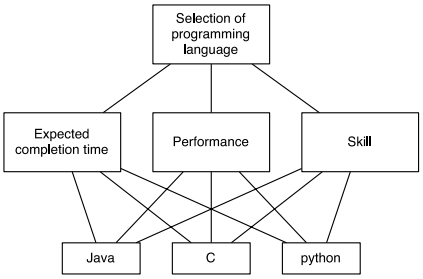

Figure 3: Hierarchy between comparison criteria for AHP

The result of comparisons are combined in an n by n matrix, M. Naturally, $M(i,i) = 1(1 \leq i \leq n)$. For the rest:

$$\forall i(1 \leq i \leq n) \forall j(1 \leq j \leq n \wedge i \neq j), M(i,j) = p_{ij}$$

The priority weighting vector E is the eigenvector of a matrix M', which is calculated from M by normalising over the columns. E is calculated by taking the average across rows of M':

$$M'(i,j) = \frac{M(i,j)}{\sum_{1 \leq k \leq n} M(i,k)}$$

$$E_i = \frac{\sum_{1 \leq k \leq n} M(k,i)}{n}$$

It is also possible to construct a hierarchy of multiple criteria [17]. To illustrate how this hierarchical AHP is achieved, suppose that AHP is used to select the programming language that will be used to implement a system. Management cares about expected completion time, performance of implemented system and programmers' skill in each language. If the candidate languages are {Java, C, Python}, then the hierarchy of criteria is as in Figure 3. First, programming languages are evaluated against each criterion in the middle level. This produces a set of priority weighting vectors, V. Second, the criteria in the middle level are prioritised, i.e., the relative importance between completion time, performance and skill level is determined using AHP. This produces another priority weighting vector, E. The final weighting vector is calculated by calculating the weighted sum of the vectors in V; for each criterion, the weight is given by E.

3.2 User Model

The approach proposed in this paper will be evaluated with respect to an 'ideal user model' and more 'realistic user' model that simulates human errors. The *ideal* user provides comparisons that are consistent with the real fault detection capability of test cases. While this may be over-optimistic, it provides an upper limit on the effectiveness of cluster-based test case prioritisation using AHP.

Since AHP allows the user to compare two entities with degrees of preference rather than simple binary relations, the ideal user model needs to consider how to quantitatively differentiate the relative importance of test cases. Previous work using human input for test case prioritisation only required binary relations, which are obtained by checking which test case detects more faults than the other. We derive varying degrees of relative importance by checking how much difference there is between the number of faults detected by two test cases.

Suppose two test cases t_A and t_B are being compared. Let n_A be the number of faults detected by t_A, and n_B by t_B. The 'ideal' user model used in the paper sets the scale of preference between t_A and t_B, p_{AB}, as shown in Table 2.

Condition	p_{AB}	Description
$n_A = n_B$	1	Equal
$n_A > 0$ and $n_B = 0$	7	Very Strongly prefer t_A
$n_A > 0, n_B > 0, n_A \geq 3n_B$	9	Extremely prefer t_A
$n_A > 0, n_B > 0, n_A \geq 2n_B$	7	Very Strongly prefer t_A
$n_A > 0, n_B > 0, n_A \geq n_B$	5	Strongly prefer t_A
$p_{BA} = \frac{1}{p_{AB}}$		

Table 2: Scale of preference for the ideal user model

For a more realistic user model, we introduce a model of human error. Suppose that test case t_A and t_B are being compared with the result of p_{AB}. Let p'_{AB} be the result with error. There are eight types of errors; Table 3 shows all eight types of error.

Errors of type 1 and 2 occur when a human tester claims that two test cases are equally important, when in fact one of them is more important than the other. Errors of type 3 and 4 occur when a human tester claims that a test case is more important than the other, when in fact it is the opposite. Errors of type 5 and 6 occur when a human tester claims that a test case is more important than the other when in fact they are equally important. Finally, errors of type 7 and 8 occur when a human tester correctly claims the inequality relation between two test cases, but answers the ratio of relative importance incorrectly. In order to only include errors that mean the human judgement is definitely wrong, only errors of type 1 to 6 are considered in the empirical studies.

Type	Original	Error
1	$p_{AB} > 1$	$p'_{AB} = 1$
2	$p_{AB} < 1$	$p'_{AB} = 1$
3	$p_{AB} > 1$	$p'_{AB} < 1$
4	$p_{AB} < 1$	$p'_{AB} > 1$
5	$p_{AB} = 1$	$p'_{AB} > 1$
6	$p_{AB} = 1$	$p'_{AB} < 1$
7	$p_{AB} > 1$	$p'_{AB} > 1$ and $p'_{AB} \neq p_{AB}$
8	$p_{AB} < 1$	$p'_{AB} < 1$ and $p'_{AB} \neq p_{AB}$

Table 3: Different types of errors.

In order to avoid any bias, we use the simplest uniformly distributed error model. Given an error rate e, which is a real number between 0 and 1, the model provides a correct answer with probability $1 - e$. With probability e, an error is introduced by changing the result of the comparison to one of alternative results defined by the types of errors with a uniformly distributed probability.

3.3 Hierarchy

AHP can deal with multiple prioritisation criteria if the user can specify relative importance between different criteria. In the proposed prioritisation technique, both a single criterion hierarchy model and multiple criteria hierarchy model are utilised. The single criterion hierarchy model requires only the comparisons from the hu-

man expert. This model allows us to observe how well the proposed technique performs when expert knowledge is used alone. However, the model may produce sub-optimal prioritisation because comparisons only reveal information about the relative quantity of faults found by test cases, not their location. For example, consider the test cases shown in Table 4. The ideal human expert will make pair-wise comparisons $t_1 \prec t_2, t_1 \prec t_3$, and $t_2 \prec t_3$. Since these comparisons are consistent with each other, the sequence provided by AHP is $< t_1, t_2, t_3 >$. However, the optimal ordering is $< t_1, t_3, t_2 >$ because of the overlap in detected faults between t_1 and t_2.

Test Case	Branch 1 (Fault 1)	Branch 2 (Fault 2)	Branch 3 (Fault 3)	Branch 4 (Fault 4)
t_1	x	x	x	
t_2	x	x		
t_3				x

Table 4: The optimal sequence is $< t_1, t_3, t_2 >$. However, perfect pair-wise comparisons will result in $t_1 \prec t_2$, $t_1 \prec t_3$, and $t_2 \prec t_3$, which will produce the sub-optimal sequence $< t_1, t_2, t_3 >$.

To overcome this limitation of AHP, the hierarchical AHP model utilises two prioritisation criteria: expert knowledge obtained from pair-wise comparisons and coverage-based prioritisation results. The user comparisons matrix is obtained using the AHP procedure described above. The structural coverage matrix reflects the result of coverage-based prioritisation. Given a result of statement coverage-based prioritisation, the coverage matrix is filled by Algorithm 3. Note that if t_i comes before t_j in statement coverage-based prioritisation, $p_{t_i t_j}$ is set to preference scale of 3, which is weaker than the values used by the user model as shown in Table 2. This is to ensure that the comparisons based on statement coverage should not override the comparisons from the human expert (i.e coverage-based comparisons are made in weaker terms than comparisons by the human expert).

Algorithm 3: Coverage Matrix Generator
Input: A set of n test cases, $T = \{t_1, \ldots, t_n\}$, and an ordered set of prioritised positions of each test case in T, $O =< i_1, \ldots, i_n >$, such that $i_j = k$ means jth test case is t_k
Output: An n by n coverage matrix, M
(1) **for** $i = 1$ **to** $i \leq n$
(2) $M[i][i] = 1.0$ //equal
(3) **for** $j = 1$ **to** $O_i - 1$
(4) $M[O_i][j] = 3$ // slightly favour t_i
(5) **for** $j = O_i + 1$ **to** $n - 1$
(6) $M[O_i][j] = \frac{1}{3}$ //slightly favour t_j

When using multiple criteria, AHP requires the human user to determine the relative importance not only between entities that are being prioritised (i.e. test cases) but also between criteria themselves (i.e. expert knowledge and statement-based prioritisation). Using Table 1, this paper applies a set of 9 different human-to-coverage preference values, $p_{[H][C]}, \{9, \ldots, 1, \ldots, \frac{1}{9}\}$. The ICP with single criterion AHP model will be denoted by ICP_S; ICP with hierarchical AHP model will be denoted by ICP_M.

4. EXPERIMENTAL SET-UP

4.1 Subjects

Table 5 shows the subject programs studied in this paper. They range from 412 to 122,169 LOC. Each program and its test suite is taken from Software Infrastructure Repository (SIR) [2].

`printtokens` and `schedule` are part of Siemens suite. SIR contains multiple test suites for these programs; four test suites are chosen randomly. `space` is an Array Description Language (ADL) interpreter that was developed by European Space Agency. From SIR, four test suites are chosen randomly. `gzip`, `sed`, `vim` and `bash` are widely used Unix programs. Only one test suite is available for these programs. Coverage data for the subject programs are generated using `gcov`, a widely used profiling tool for `gcc`.

Program	Test Suite	(Avg.) TS Size	LOC
`printtokens`	4	317.00	726
`schedule`	4	225.25	412
`space`	4	158.50	6,199
`gzip`	1	212	5,680
`sed`	1	370	14,427
`vim`	1	975	122,169
`bash`	1	1061	59,846

Table 5: Subject programs and their test suite size

4.2 Suitability Test Configuration

SIR contains versions with injected faults and the mapping from test cases to the faults detected by each test case; for `gzip`, `sed`, `vim` and `bash`, it contains multiple consecutive versions of the source code and corresponding fault detection information. Whenever available, the empirical study for the suitability testing utilises two distinct, consecutive versions of the program. The faults from the first version are used as AR (Already Revealed), whereas the faults from the second version is used as TBR (To Be Revealed). For `printtokens`, `schedule` and `space`, multiple versions of the source code are available but there exists a single set of fault detection information for a single version. For these programs, we randomly divide the known faults into the AR and TBR sets and re-use the structural coverage recorded from the source code of the same version. Table 6 shows the size of group AR and TBR for each program respectively.

Program	Size of AR	Size of TBR	Mult. Ver.
`printtokens`	3	4	No
`schedule`	4	5	No
`space`	18	20	No
`gzip`	2	3	Yes
`sed`	6	4	Yes
`vim`	4	3	Yes
`bash`	4	9	Yes

Table 6: Subject programs and the size of AR (Already Revealed) and TBR (To Be Revealed) sets

4.3 Evaluation

The results of ICP are compared to the optimal ordering, OP, and statement coverage-based ordering, SC. The optimal ordering is obtained by prioritising the test cases based on the fault detection information. It is impossible to know fault detection record in advance, and therefore, the optimal ordering is not available in the real world. Statement coverage-based ordering is obtained by performing additional-statement prioritisation. This method has been widely studied and known to produce reasonable results [6, 13].

In the first set of empirical studies, we evaluate ICP against OP and SC using Average Percentage of Fault Detection (APFD) metric [5]. Let T be the test suite containing n test cases and let F be the set of m faults revealed by T. For ordering T', let TF_i be the order of the first test case that reveals the ith fault. APFD value for T' is calculated as following:

$$APFD = 1 - \frac{TF_1 + \ldots + TF_m}{nm} + \frac{1}{2n}$$

ICP experiments are performed with different values for user error rates ranging from 0.05 (user is wrong 5 times out of 100 on average) to 1.0 (user is always wrong) in steps of 0.05.

4.4 Research Questions

This paper considers the following research questions:

RQ1. Effectiveness: Is there any difference in effectiveness between ICP, optimal ordering and coverage-based prioritisation?

RQ2. Configuration: When the human input is combined with other prioritisation criteria such as structural coverage, what is the ideal configuration between human input and other criteria?

In order to answer **RQ1** and **RQ2**, we evaluate two different hybrid-ICP instances, one with single hierarchy model AHP and the other with multiple hierarchy model AHP. For **RQ1**, we measure the APFD of the produced ordering and compare it to those of optimal ordering and statement coverage-based prioritisation. For **RQ2**, we execute the hybrid ICP with multiple hierarchy model AHP. The secondary prioritisation criteria is derived from statement coverage-based prioritisation using the algorithm shown in Section 3.3. We measure APFD of the test case sequences obtained by using different values of $p_{[H][C]}$ and compare these to those of optimal ordering and statement coverage-based prioritisaiton. **RQ1** and **RQ2** are answered in Section 5.1.

The second part of the empirical studies deals with tolerance and suitability. Since the quality of the results produced by the proposed technique depends directly on the quality of the human user's input, it is necessary to study how high the allowable level of error rate can be while producing results that are better than coverage-based techniques.

RQ3. Tolerance: What is the highest human error rate that can be tolerated while yielding performance superior to the coverage-based techniques?

For suitability study, we apply the automated suitability test to subject programs and their test suites using the AR faults and see if the difference between ICP and coverage-based prioritisation is consistent with the result of effectiveness and efficiency studies.

RQ4. Suitability: How accurately does the automated suitability test predict the successful result of ICP?

RQ3 is answered in the second part of the empirical study, by increasing the error rate and observing the statistics of the results. **RQ4** is answered by performing automated suitability test and comparing the results to those of effectiveness and efficiency study. **RQ3** and **RQ4** are answered in Section 5.2.

5. RESULTS AND ANALYSIS

5.1 Effectiveness & Configuration

Table 7 shows the APFD values measured from the single-hierarchy model approach, ICP_S. The cells with grey background denote configurations that outperformed statement coverage-based prioritisation in terms of APFD metric. The proposed technique only outperforms coverage-based techniques with the right combination of program and test suite. For example, suite 1 and suite 2 of `space` show improvement over SC, but suite 3 and suite 4 do not. The clustering also has detrimental effect on the test suite of `bash`; the APFD value produced by ICP_S is lower than that of SC. Overall, out of 16 prioritisation problems, ICP_S produces higher APFD than statement coverage-based prioritisation for 9 cases. Note that these results assume the human input from the *ideal* user and, therefore, the results are deterministic. The increases in APFD range from 0.5% to 21.8% with average increase of 6.5%. This provides an answer to **RQ1**.

Table 8 shows the results from the multiple-hierarchy model approaches, ICP_M. Each configuration uses different $p_{[H][C]}$ value to prioritise the criteria, i.e., comparisons made by human expert and statement coverage-based prioritisation. One trend observed in every prioritisation is that higher $p_{[H][C]}$ tends to produce higher APFD metric values. With a few exceptions, observed APFD values monotonically decrease as $p_{[H][C]}$ decreases. This implies that the ideal configuration for the hybrid ICP_M approach is to set $p_{[H][C]} = 9$, i.e. to favour the human judgement extremely. This provides an answer to **RQ2**. With $p_{[H][C]} = 9$, the increases in APFD range from 0.7% to 22% with average increase of 6.4%.

5.2 Tolerance & Suitability

Now we turn to the second set of research questions. Regarding the tolerance study and **RQ3**, Figure 4 and Figure 5 show how APFD values from ICP_S and ICP_S deteriorate as error rate increases from 0.05 to 1.0. Comparing Figure 4 and Figure 5 with Table 7 and Table 8, test suites for which ICP does not achieve improvement tend to be less resistant to increasing error rate. On the other hand, test suites for which ICP is capable of making improvement over statement coverage-based prioritisation tend to produce more robust APFD values as the error rate increases. The test suite for `gzip` is capable of producing higher APFD values than statement coverage with error rates up to 0.45. Surprisingly, test suites for `schedule`, `sed`, `vim`, and two test suites for `space` are capable of producing higher APFD values than statement coverage with error rates up to 1.0, i.e. under the situation when the human expert always makes incorrect comparisons.

Figure 6 provides an explanation to this seemingly counter-intuitive phenomenon. Figure 6 contains following three boxplots. In each subplot, *Random* shows APFD of random ordering of test cases with no clustering; *RCRP* (Random Clustering Random Prioritisation) shows APFD of randomised ICP with random clustering with $k = 14$ and *HCRP* (Hierarchical Clustering Random Prioritisation) shows APFD of randomised ICP with hierarchical clustering. The solid horizontal line represents APFD value of the optimal ordering; the dotted horizontal line represents APFD value of the statement coverage-based prioritisation. There exists a common trend between all programs for which ICP produced successful improvement. For these programs, either the mean of *HCRP* or the upper quartile observation of the box plot is higher than the APFD of statement coverage-based prioritisation. It can be concluded that the clustering with $k = 14$ works for the prioritisation of these programs and their test suites. Note that all the prioritisation is performed randomly for *HCRP*. Our conjecture is that, for these programs, any prioritisation technique that performs better than a purely random approach will eventually make an improvement over statement coverage-based prioritisation.

Table 9 confirms this conjecture, and provides an answer for **RQ4**. It compares the result of the suitability test for faults in AR and TBR sets of each program with the optimal ordering and the statement coverage-based prioritisation. Hierarchical Clustering/Statement Prioritisation (HCSP) represents the ICP with statement coverage prioritisation both for intra- and inter-cluster stage, combined with hierarchical agglomerative clustering of $k = 14$. No Clustering/Statement Prioritisation (NCSP) represents traditional statement coverage-based prioritisation. Note that both configurations are deterministic and can be automated. For the faults in the AR set, both NCSP and HCSP are executed. If the result of

Subject	printtokens				schedule				space				gzip	sed	vim	bash
Test Suite	1	2	3	4	1	2	3	4	1	2	3	4				
OP	0.995	0.995	0.997	0.995	0.991	0.995	0.993	0.993	0.983	0.985	0.985	0.982	0.996	0.997	0.998	0.999
ICP_S	0.806	0.974	0.967	0.868	0.824	0.917	0.952	0.913	0.948	0.933	0.930	0.927	0.996	0.905	0.903	0.144
SC	0.930	0.992	0.972	0.960	0.806	0.865	0.782	0.844	0.899	0.863	0.948	0.947	0.980	0.876	0.899	0.210

Table 7: APFD values obtained from ICP_S and ideal user model compared to those of the optimal ordering and the statement coverage-based prioritisation. Cells with grey background represent the fact that ICP_S outperformed statement coverage-based prioritisation in terms of APFD.

Subject		printtokens				schedule				space				gzip	sed	vim	bash
Test Suite		1	2	3	4	1	2	3	4	1	2	3	4				
OP		0.995	0.995	0.997	0.995	0.991	0.995	0.993	0.993	0.983	0.9852	0.985	0.982	0.996	0.997	0.998	0.999
	9	0.807	0.974	0.967	0.871	0.825	0.916	0.954	0.912	0.946	0.938	0.931	0.927	0.996	0.905	0.905	0.144
	7	0.807	0.974	0.967	0.871	0.825	0.916	0.954	0.912	0.946	0.939	0.931	0.926	0.996	0.905	0.905	0.144
	5	0.807	0.974	0.967	0.871	0.825	0.915	0.954	0.912	0.947	0.939	0.931	0.926	0.996	0.905	0.905	0.144
ICP_M	3	0.807	0.974	0.966	0.871	0.825	0.914	0.952	0.912	0.946	0.940	0.931	0.926	0.996	0.905	0.905	0.144
$p_{[H][C]}$	1	0.807	0.974	0.966	0.871	0.824	0.915	0.951	0.912	0.946	0.939	0.930	0.923	0.996	0.905	0.904	0.144
	1/3	0.808	0.973	0.966	0.870	0.823	0.905	0.945	0.909	0.943	0.937	0.929	0.920	0.991	0.902	0.904	0.144
	1/5	0.807	0.973	0.966	0.870	0.820	0.903	0.943	0.907	0.943	0.936	0.928	0.920	0.988	0.902	0.904	0.144
	1/7	0.807	0.973	0.966	0.870	0.821	0.901	0.941	0.906	0.943	0.936	0.928	0.919	0.987	0.902	0.904	0.144
	1/9	0.806	0.973	0.966	0.870	0.821	0.901	0.941	0.904	0.943	0.935	0.928	0.919	0.985	0.902	0.904	0.144
SC		0.930	0.992	0.972	0.960	0.806	0.865	0.782	0.844	0.899	0.863	0.948	0.947	0.980	0.876	0.899	0.210

Table 8: APFD values obtained from ICP_M with different $p_{[H][C]}$ values and ideal user model, ranging from 'extremely favours human expert's judgement (9)' to 'extremely favours coverage-based prioritisation ($\frac{1}{9}$)'.

HCSP is equal to or higher than that of NCSP, the specific pair of SUT and test suite is said to *pass* the suitability test. A *pass* means that the hierarchical clustering has a positive impact on statement coverage-based prioritisation. Therefore, we expect that replacing statement coverage-based prioritisation with techniques such as AHP will only improve the prioritisation of any SUT and test suite that passed suitability test. This expectation is then checked using the TBR set of faults. If ICP_M produces higher APFD than NCSP for a pair of (SUT, test suite) that passed suitability test, it can be said that the test produced a correct prediction. Since the aim of the suitability test is to avoid wasting human effort, false positive tests presents higher risk than false negative tests.

The results in Table 9 indicate the passed tests with grey background. For all 8 tests that passed, the subsequent experiments with faults in the TBR set confirm the result of suitability test. That is, ICP_M produces higher APFD than statement coverage-based prioritisation. There are two false negative results, marked with (*). Suite 2 of `schedule` and `vim` do not pass the suitability test, but ICP does produce higher APFD than NCSP. There is no false positive test result. The remaining 6 pairs of program/test suite do not pass the suitability test, and the subsequent experiments with faults in the TBR set confirm the correctness of the suitability test. In summary, **RQ4** is answered positively with 14 correct predictions (8 pass, 6 fail) out of 16 cases. The remaining two cases are comparatively harmless false negatives.

5.3 Limitations & Threats to Validity

Threats to internal validity concern the factors that might have affected the performance the proposed technique. The accuracy of execution trace data and fault detection data can have a significant impact on the performance. To address this, the execution traces were obtained using a widely used and well known compile/profiling tool, `gcc` and `gcov`. Fault detection data were obtained from SIR [2].

Threats to external validity concern the conditions that limit generalisation of the results. The novelty of the proposed technique lies in the combination of clustering and pair-wise comparisons. This paper uses the agglomerative hierarchical clustering for the former and AHP for the latter, but other combinations of techniques may produce different results. Different combinations of techniques should be studied in order to address this concern. The representativeness of the test suites and subject programs is another primary concern. However, the paper studies programs with sizes ranging from 412 LoC to over 100KLoc. When multiple test suites are available, four different test suites were randomly chosen to avoid any bias based on the choice of test suites.

6. RELATED WORK

Test case prioritisation techniques aim to improve the efficiency of regression testing by prioritising test cases so that earliest fault detection is possible. Since fault detection information is not known in advance, test case prioritisation techniques tend to use surrogates as their criteria. Structural coverage is an often used surrogate for fault detection capability [6, 7, 11, 12, 14, 15, 16, 20].

Rothermel et al. performed empirical studies of several different prioritisation techniques [15, 16]. The authors consider diverse surrogates such as branch-total, branch-additional, statement-total, statement-additional, Fault-Exposing Potential (FEP)-total, and FEP-additional. The 'total' approaches correspond to the normal greedy algorithm, whereas 'additional' approaches correspond to the additional greedy algorithm. The authors report that there is no single criterion that dominates others in terms of rate of fault detection. Elbaum et al. extended the additional greedy approach of Rothermel et al. by incorporating the fault criticality and the cost of executing test cases [7].

The use of clustering in test case prioritisation is not entirely new. Leon et al. prioritised test cases by forming clusters of their execution profile [10] and showed that their approach can outperform coverage-based prioritisation. However, this work differs in its aim (to reduce the cost of human based pair-wise comparison) and the choice of dissimilarity metric for clustering (hamming distance compared to Euclidean distance of Leon et al.).

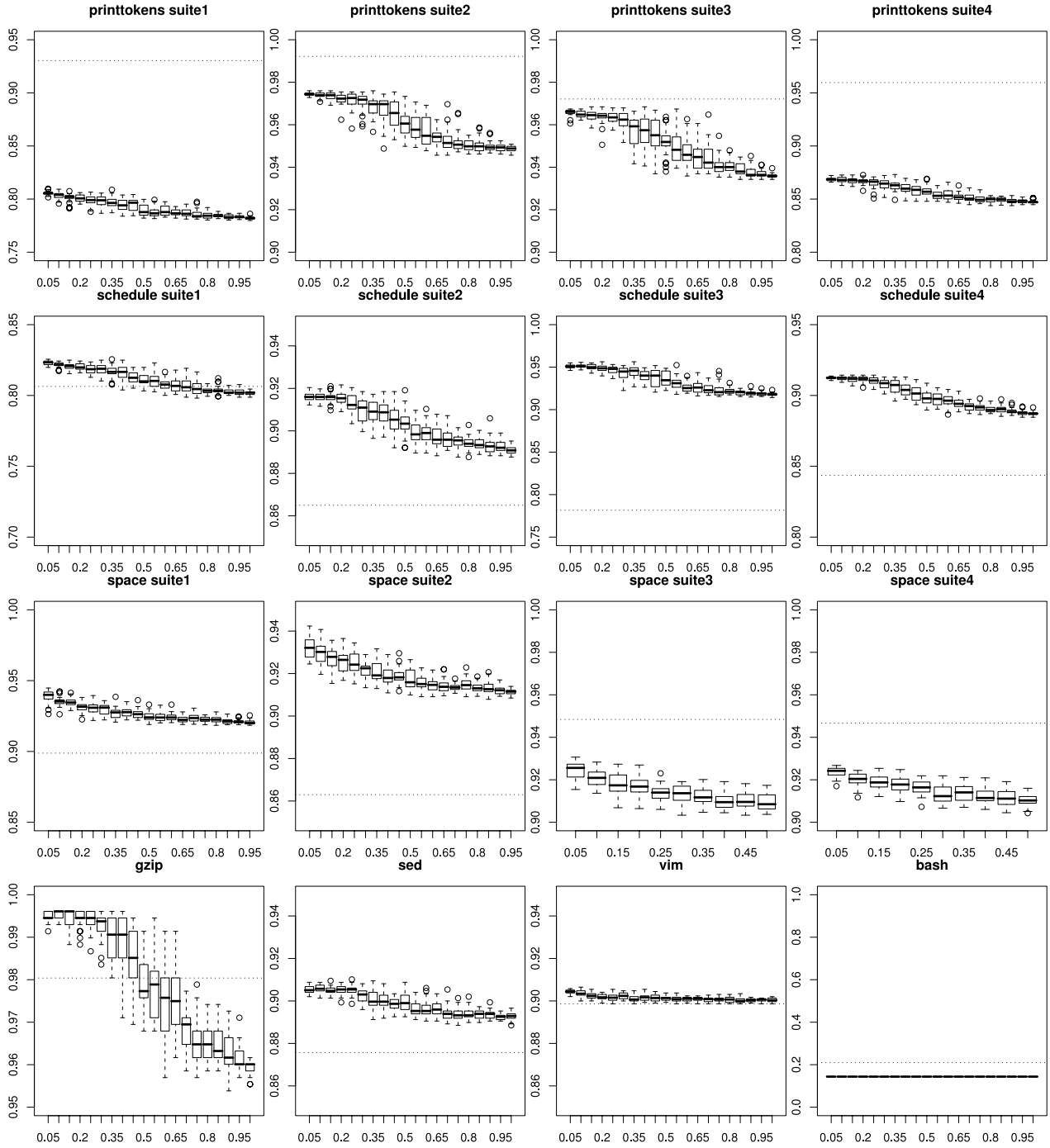

Figure 4: Boxplots of APFD values of ICP_S configuration with error rate ranging from 0.05 to 1.0 in steps of 0.05 on the x-axis. The y-axis shows the observed APFD metric values. For each error rate value, experiments are repeated 30 times to cater for the randomness in the error model. The horizontal dotted line shows the APFD value of statement coverage-based prioritisation. Surprisingly, for the test suites for which ICP_S showed an improvement in Table 7, the mean APFD values tend to stay above this dotted line, even when the error rate is above 0.5. In fact, with the exception of `schedule1` and `gzip`, even the error rate of 1.0 produces successful results for the test suites for which ICP_S showed an improvement in Table 7.

AHP has been widely adopted in various software engineering fields where the only meaningful prioritisation criteria is often the human preference. Karlsson et al. applied AHP to requirement prioritisation and empirically compared AHP with several different techniques [9]. Finnie et al. applied AHP to project management and prioritised productivity factors in software development [8]. Douligeris et al. applied AHP to evaluate the Quality-of-Service (QoS) in telecommunication networks [4].

Previous work in the test case management field studied how human involvement can improve the quality of test case prioritisation. Tonella et al. have successfully applied the Case-Base Ranking (CBR) machine learning technique to test case prioritisation [19].

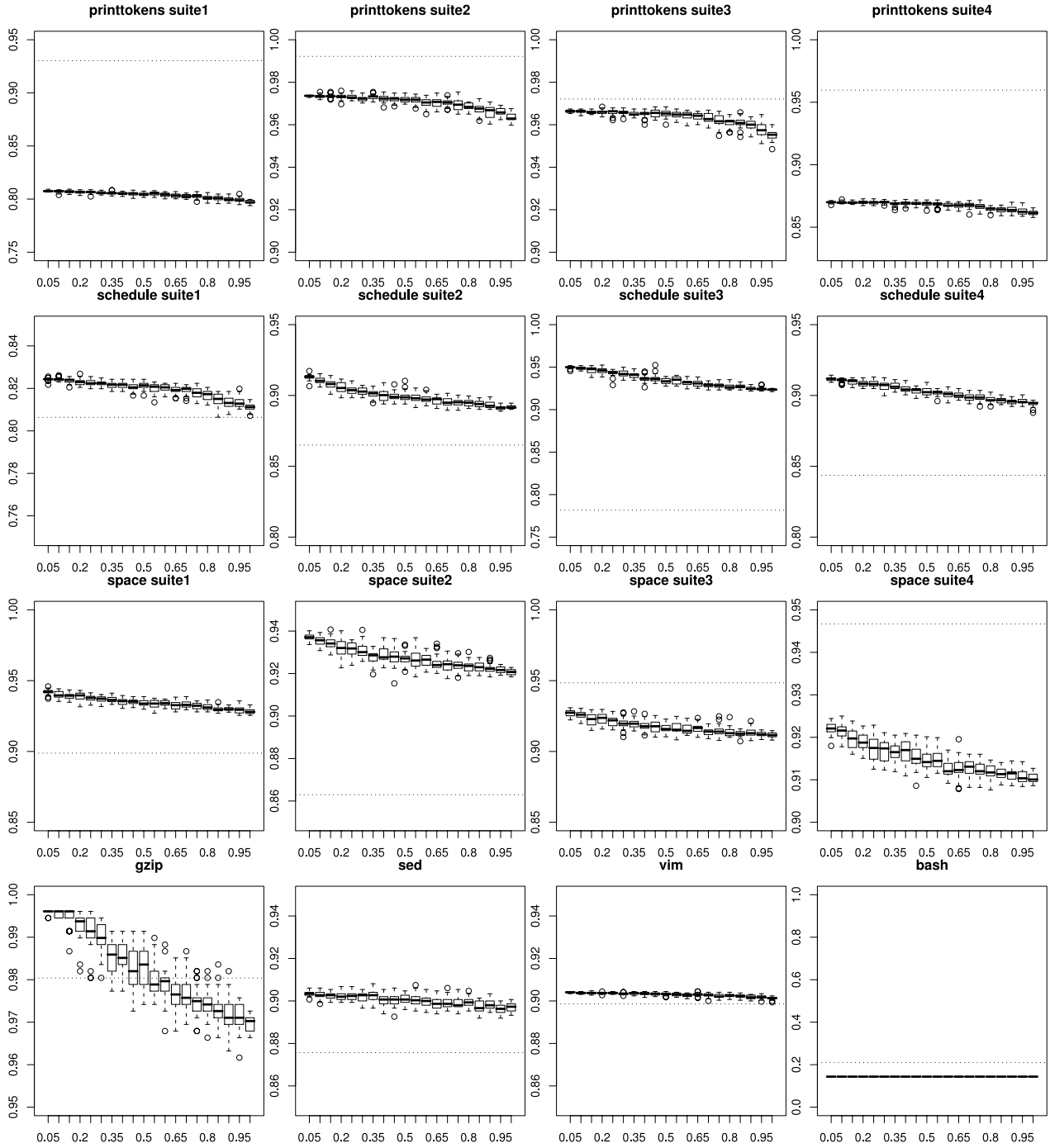

Figure 5: Boxplots of APFD values of ICP_M configuration with error rate ranging from 0.05 to 1.0 in steps of 0.05 on the x-axis. The y-axis shows the observed APFD metric values. The trend observed in Figure 4 continues. However, it can also be observed that the secondary prioritisation criteria (statement coverage) compliments human input. APFD values show smaller variances compared to Figure 4 and, in some cases, more tolerance in the presence of human error, for example, the case with test suite 1 of `schedule`.

CBR tries to learn the ordering between the test cases from the additional information it is given such as structural coverage. During the process, CBR presents the human tester with a pair of test cases and asks which one is more important. It combines the human input with the information originally available, and produces an ordering of test cases. The empirical results show that prioritisation using CBR can outperform the existing coverage-based prioritisation techniques.

While this paper starts from the same assumption (that human involvement can improve test case prioritisation), there are several significant differences from this previous work. First, this paper introduces clustering to deal with the high cost of AHP. Without the reduction in effort, the cost of AHP has been considered inhibitive. Second, this paper introduces the error rate in the user model, and thereby provides more realistic observation of the performance of the technique. Finally, the paper introduces an automated assess-

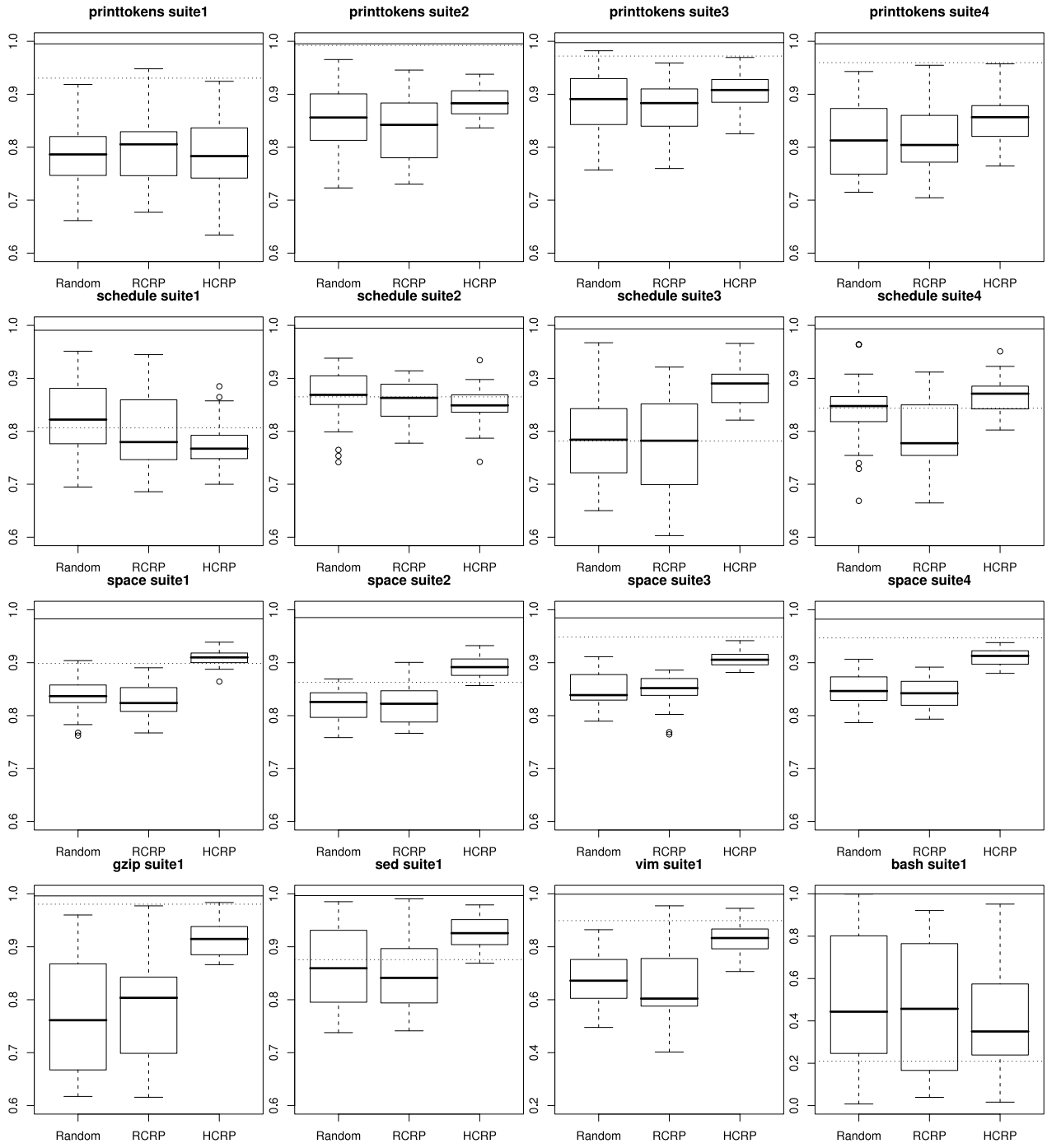

Figure 6: Boxplots of random prioritisation results. On the x-axis, 'Random' represents random prioritisation with no clustering; 'RCRP' represents random clustering and random ICP; 'HCRP' represents hierarchical clustering and random ICP. The y-axis shows the observed APFD metric values. For programs for which ICP performed well in Section 5.1, HCRP partially outperforms statement coverage-based prioritisation (represented by the dotted lines) even with random prioritisation.

ment technique that determines whether human effort will be justified by the expected results.

7. CONCLUSION & FUTURE WORK

This paper introduced the use of clustering for human interactive test case prioritisation. Human interaction is achieved by using AHP, which is a widely used decision making tool that has been previously adopted by the Requirements Engineering community. Clustering is applied to reduce the number of pair-wise comparisons required by AHP, making it scalable to regression testing problems. The paper introduced a hybrid Interleaved Clusters Prioritisation technique to combine these two techniques. The empirical studies show that the hybrid ICP algorithm can outperform the traditional coverage-based prioritisation for some programs, even when the human input is erroneous. The paper also presents an

Subject	printtokens				schedule				space				gzip	sed	vim	bash
Test Suite	1	2	3	4	1	2	3	4	1	2	3	4				
OP	0.995	0.995	0.998	0.995	0.991	0.995	0.993	0.993	0.983	0.985	0.985	0.982	0.996	0.997	0.998	0.999
NCSP AR	0.936	0.997	0.998	0.965	0.899	0.974	0.922	0.949	0.958	0.957	0.963	0.974	0.817	0.958	0.946	0.804
HCSP AR	0.736	0.976	0.998	0.896	0.984	0.970*	0.972	0.986	0.964	0.977	0.941	0.939	0.831	0.959	0.890*	0.746
NCSP TBR	0.915	0.993	0.953	0.949	0.831	0.880	0.854	0.883	0.918	0.914	0.962	0.973	0.980	0.876	0.899	0.210
ICP_M TBR	0.755	0.954	0.978	0.875	0.994	0.992	0.992	0.992	0.966	0.982	0.956	0.944	0.996	0.905	0.905	0.144

Table 9: Results of the suitability test. NCSP is the traditional statement coverage-based prioritisation. HCSP is ICP with statement coverage prioritisation for both intra- and inter-cluster prioritisation. If HCSP performs no worse than NCSP, the test passes, i.e. ICP is expected to outperform NCSP with the faults in the TBR set. Cells with grey background show passed tests with correct prediction; cells with white background denote failed tests with correct prediction. The second test suite of `schedule` and the test suite of `vim` produce false negative results (denoted by *).

automated suitability test that can accurately predict whether the hybrid ICP will make a positive improvement to the prioritisation of a specific pair of SUT and test suite, ensuring that no human effort is wasted on regression testing scenarios that are inappropriate for the technique. Future work will consider different clustering criteria in order to produce closer approximation to the semantics of fault detection.

8. REFERENCES

[1] P. Avesani, C. Bazzanella, A. Perini, and A. Susi. Facing scalability issues in requirements prioritization with machine learning techniques. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 297–306, Paris, France, August 2005.

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

[3] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Transactions on Software Engineering*, 32(9):733–752, 2006.

[4] C. Douligeris and I. Pereira. A telecommunications quality study using the analytic hierarchy process. *IEEE Journal on Selected Areas in Communications*, 12(2):241–250, Feb 1994.

[5] S. Elbaum, A. Malishevsky, and G. Rothermel. Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, Feb 2002.

[6] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *In Proceedings of the 2nd International Symposium on Software Testing and Analysis*, pages 102–112, Portland, USA, 2000.

[7] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 329–338, Toronto, Canada, May 2001.

[8] G. R. Finnie, G. Wittig, and D. I. Petkov. Prioritizing software development productivity factors using the analytic hierarchy process. *The Journal of Systems and Software*, 22(2):129–139, August 1993.

[9] J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for prioritizing software requirements. *Information & Software Technology*, 39(14-15):939–947, 1998.

[10] D. Leon and A. Podgurski. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering*, pages pp. 442–456, 2003.

[11] 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.

[12] A. Malishevsky, G. Rothermel, and S. Elbaum. Modeling the cost-benefits tradeoffs for regression testing techniques. In *Proceedings of the 18th International Conference on Software Maintenance*, pages 230–240, Montreal, Canada, October 2002.

[13] A. G. Malishevsky, J. R. Ruthruff, G. Rothermel, and S. Elbaum. Cost-cognizant test case prioritization. Technical Report TR-UNL-CSE-2006-0004, Department of Computer Science and Engineering, University of Nebraska-Lincoln, March 2006.

[14] G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri, and B. Davia. The impact of test suite granularity on the cost-effectiveness of regression testing. In *Proceedings of the 24th International Conference on Software Engineering*, pages 130–140, New York, NY, USA, May 2002.

[15] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: An empirical study. In *Proceedings of the 15th International Conference on Software Maintenance*, pages 179–188, Los Alamitos, California, USA, 1999.

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

[17] T. Saaty. *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, New York, USA, 1980.

[18] H. Srikanth, L. Williams, and J. Osborne. System test case prioritization of new and regression test cases. In *Proceedings of International Symposium on Empirical Software Engineering*, pages 64–73, November 2005.

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

[20] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time aware test suite prioritization. In *ISSTA '06: Proceedings of the 2006 International Symposium on Software Testing and Analysis*, pages 1–12, New York, NY, USA, 2006. ACM Press.