Reducing the cost of applying adaptive test cases

R. M. Hierons [a] H. Ural [b]

[a] *School of Information Systems, Computing, and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH.*

[b] *School of Information Technology and Engineering, Faculty of Engineering, University of Ottawa, 800 King Edward Avenue, Ottawa, Ontario, K1N 6N5, Canada*

keywords: State-based testing; Test Execution; Adaptive test cases; Minimising Cost.

Abstract

The testing of a state-based system may involve the application of a number of adaptive test cases. Where the implementation under test (IUT) is deterministic, the response of the IUT to some adaptive test case γ_1 could be capable of determining the response of the IUT to another adaptive test case γ_2. Thus, the expected cost of applying a set of adaptive test cases depends upon the order in which they are applied. This paper explores properties of adaptive test cases and considers the problem of finding an order of application of the elements from some set of adaptive test cases, which minimises the expected cost of testing.

1 Introduction

There are many approaches that generate tests on the basis of some source of information, such as the specification or the structure of the code. Where a system is state-based, testing involves the application of sequences of input values. Such sequences are called test sequences. A test sequence, or a set of test sequences, may represent some test purpose.

An adaptive test case is applied in an adaptive experiment which is a process in which at each stage the input applied depends upon the input/output sequence that has been observed. Once the adaptive test case has finished the system is returned to its initial state, using some postamble, and is ready for the next adaptive test case to be applied.

The use of adaptive tests cases is often essential where the specification is either nondeterministic or incomplete and may be useful if we wish to have robust tests; tests that may achieve their test purpose (such as testing a particular transition) even if there is an earlier failure. Adaptive test cases have been used for testing from a (possibly nondeterministic) finite state machine [1,9,10,15,21,23], testing from a general state-based specification [15], and generally in protocol conformance testing [13]. Algorithms for testing from a specification written in a process algebra such as LOTOS [3] typically produce adaptive test cases with verdicts [20]. Adaptivity is a core element of the European Telecommunications Standards Institute (ETSI) test description language TTCN (see, for example, [22]).

Among the motivation and potential uses of adaptive test cases we may also mention the following: there are systems developed based on requirements that provide a variety of options for the implementers such as standardized protocols. A test suite constructed from such a requirement typically consists of adaptive test cases where the expected behaviour is represented in the form of trees rather than sequences. For example, a suite of tests in TTCN contains a behaviour tree for each test which is a succinct representation of an adaptive test case. Each leaf of the behaviour tree is associated with a verdict which represents one of the test results, namely, pass, fail, and inconclusive. Such a set of tests can be constructed manually or automatically based on a set of test purposes or in a random manner. Further, a third party set of tests represented in TTCN can be purchased. Thus, it is important to identify an order of application of the adaptive test cases represented as trees in TTCN in order to reduce the test effort.

Since testing is extremely expensive, reductions in the cost of testing could lead to cheaper software and might allow more rigorous testing to be applied. The test process contains several components, including the generation of a test suite, the application of the test cases from this test suite, and checking the behaviour observed in testing against the requirements. This paper investigates ways in which the cost of *applying* adaptive test cases may be reduced. The cost of applying adaptive test cases is particularly important where the use of a test cases requires expensive/scarce equipment or facilities, involves human participation, requires a system to be reconfigured, or takes a significant time to complete. Crucially, the results given in this paper show how the expected cost of applying a set of adaptive test cases may be reduced without diminishing their effectiveness.

When the *implementation under test (IUT)* is deterministic, the input/output sequence produced in response to one adaptive test case γ_1 may be capable of providing information that fully decides the response of the IUT to some other adaptive test case γ_2 [11]. Where this is the case, using γ_1 before γ_2 may reduce the expected test execution effort. Note that the response of the IUT

to γ_1 might be capable of determining the response of the IUT to γ_2 but not be guaranteed to do this. In such cases we cannot simply eliminate γ_2 from the test suite without risking reducing the effectiveness of this test suite. Even if the response to γ_1 can determine the response to γ_2, relationships involving other adaptive test cases may affect the best relative ordering of γ_1 and γ_2. This paper considers the problem of finding an order of execution, of a given set of adaptive test cases, that minimises the expected cost of executing the tests. Such an ordering is considered to be an optimal ordering.

Interestingly, related problems have been considered in the regression testing of stateless systems. Work in this area has considered ways of reducing the number of test cases used and the problem of finding an optimal order in which to execute test cases (see, for example, [8,17,18]). However, since the systems are stateless, the tests are individual inputs rather than adaptive test cases and thus the problems are quite different. Further, some of the algorithms applied to reduce the cost of regression testing may also lead to less effective regression testing [18]. In contrast, this paper considers algorithms that reduce the expected cost of testing while *guaranteeing* to preserve the inherent effectiveness of the original set of adaptive test cases.

This paper [1] describes how it is possible to reduce the expected cost of the application of adaptive test cases. We introduce an algorithm, for the application of adaptive test cases, that allows adaptive test cases to be deleted where, due to previously observed input/output sequences, they are no longer required. We show that the order of application of a set of adaptive test cases affects the expected cost of testing and formalise this. We prove that the problem of finding an optimal order of application is NP-hard and give two procedures for reducing the scale of the optimisation problem. We then give a polynomial time algorithm that, under certain well-defined conditions, is guaranteed to return the optimal ordering. This algorithm is extended to form a heuristic for the general case.

This paper extends [11] in a number of ways. First, we give an explicit algorithm for test execution; this allows potential savings to be utilised. Further, we prove that the problem of finding an optimal ordering is NP-hard. We also give, and prove, the (polynomial time) complexity of the algorithms and functions defined in this paper. Finally, we extend the algorithm, for choosing an ordering of the adaptive test cases, to the general case.

This paper is structured as follows. Section 2 describes test sequences, directed graphs, and adaptive test cases. Section 3 then considers conditions under which the relative ordering of adaptive test cases may be significant. In

[1] This paper is an extension of [11], which won the Best Paper award at the 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2003).

Section 4 we prove that the overall optimisation problem is NP-hard. Section 5 considers two ways in which the optimisation problem may be simplified: by merging adaptive test cases and by dividing the optimisation problem into a set of optimisation problems. Section 6 introduces a polynomial time algorithm that generates the optimal ordering under a well-defined condition. It then extends this to a general heuristic. Finally, Section 7 draws conclusions and discusses future work.

2 Preliminaries

2.1 Sequences

Throughout this paper X and Y will denote the finite input and output domains of the IUT. Given a set A, A^* will denote the set of sequences of elements from A, including the empty sequence ϵ.

It will be assumed that the IUT is a black box state-based system whose functional behaviour is being tested. Thus testing leads to the observation of input/output sequences of the form $\langle x_1/y_1, \ldots, x_k/y_k\rangle \in (X/Y)^*$ where $x_1, \ldots, x_k \in X$ and $y_1, \ldots, y_k \in Y$ and y_i may be a tuple of outputs ($1 \leq i \leq k$). A preset test sequence is some element of X^*.

For convenience, the input/output sequence $\langle x_1/y_1, \ldots, x_k/y_k\rangle$ may be represented as I/O where $I = \langle x_1, \ldots, x_k\rangle$ and $O = \langle y_1, \ldots, y_k\rangle$. Given sequences c and d, cd will denote the result of concatenating c and d. For example, $\langle a/0\rangle\langle a/1, b/0\rangle = \langle a/0, a/1, b/0\rangle$. Given sets C and D of sequences, CD will denote the set formed by concatenating the elements of C with the elements of D. Thus $CD = \{cd | c \in C \wedge d \in D\}$.

Given a sequence $b \in A^*$, $pre(b)$ will denote the set of prefixes of b. Given a set B of sequences $Pre(B)$ will denote the set of prefixes of sequences from B. These are defined more formally by the following.

Definition 1 *Given a sequence $b \in A^*$ and a set $B \subseteq A^*$:*

$$pre(b) = \{b' \in A^* | \exists b'' \in A^*.b = b'b''\}$$

$$Pre(B) = \bigcup_{b \in B} pre(b)$$

When a set of test sequences, or adaptive test cases, is applied, the IUT is returned to its initial state after each test using a test postamble [12]. This ensures that each test is applied in the initial state of the IUT.

2.2 Directed Graphs

A directed graph is a set of vertices with arcs between them.

Definition 2 *A* directed graph (digraph) G *is defined by a pair* (V, E) *in which* $V = \{v_1, \ldots, v_n\}$ *is a finite set of vertices and* $E \subseteq V \times V$ *is a set of directed edges between the vertices of* G*. An element* $e = (v_i, v_j) \in E$ *represents an edge from* v_i *to* v_j*. Given an edge* $e = (v_i, v_j)$*,* $start(e)$ *denotes* v_i *and* $end(e)$ *denotes* v_j*.*

Definition 3 *Given a digraph* $G = (V, E)$ *and a vertex* $v \in V$*,* $indegree_E(v)$ *denotes the number of edges from* E *that end in* v *and* $outdegree_E(v)$ *denotes the number of edges from* E *that start at* v*. These are defined by the following:*

$$indegree_E(v) = |\{(v_i, v_j) \in E | v_j = v\}|$$

$$outdegree_E(v) = |\{(v_i, v_j) \in E | v_i = v\}|$$

.

Given a digraph $G = (V, E)$, a non-empty sequence $\langle e_1, \ldots, e_m \rangle$ of edges from E, in which for all $1 \leq k < m$ $end(e_k) = start(e_{k+1})$, is a *path* from $start(e_1)$ to $end(e_m)$ in G. A path is a *cycle* if its initial and final vertices are the same and no other vertex is repeated. A digraph is said to be *acyclic* if it has no cycles.

Given a digraph G, the following defines the result of removing one or more vertices from G.

Definition 4 *Let* $G = (V, E)$ *be a digraph and* $V' \subseteq V$*. Then* $G \setminus V'$ *denotes the digraph formed by removing every vertex contained in* V'*, and the associated edges, from* G*:*

$$G \setminus V' = (V \setminus V', E \setminus \{(v_i, v_j) | v_i \in V' \vee v_j \in V'\})$$

2.3 Adaptive test cases

Testing typically involves applying input sequences to the IUT and observing the output produced. Sometimes the input sequences used are preset: each is fully determined before it is applied. However, a test may be adaptive: the next input provided in a sequence may depend on the output produced in response to the previous input values. Such a test is called an *adaptive test*

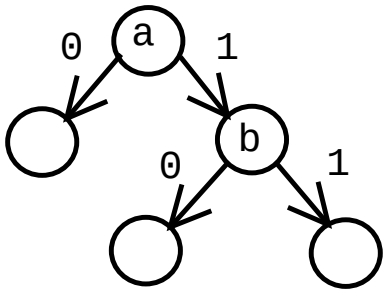

Fig. 1. An adaptive test case

case. Adaptive test cases are used in a number of areas including protocol conformance testing (see, for example, [1,13,21]).

In this paper $\mathcal{T}$ will denote the set of all adaptive test cases that may be formed using finite input domain X and finite output domain Y. The set $\mathcal{T}$ may be defined recursively in the following manner.

Definition 5 *Each element $\gamma \in \mathcal{T}$ is one of:*

- *null*
- *a pair (x, f) in which $x \in X$ and f is a function from Y to $\mathcal{T}$.*

An adaptive test case γ is applied in the following manner. If $\gamma = null$ then the adaptive test case ends. If $\gamma = (x, f)$ then the input x is applied and the output y is observed. The adaptive test case $f(y)$ is then applied. Note that in practice, it is more efficient to allow a function f used in defining an adaptive test case to be a partial function, where execution terminates if it is to be applied to an output y such that $y \notin dom\ f$. However, in order to simplify the exposition we will assume that any function used in an adaptive test case is total.

By definition, all adaptive test cases are finite: their application must always terminate. An adaptive test case may be represented by a tree. For example, the tree in Figure 1 represents an adaptive test case in which the first input is a, no further input is provided if the output is 0, and the input b is provided if the output is 1. Whatever the response to b after a, the adaptive test case then terminates. The adaptive test case γ given in Figure 1 may be defined in the following way: $\gamma = (a, f)$, $f(0) = null$, $f(1) = (b, f')$, $f'(0) = null$, and $f'(1) = null$.

Given an adaptive test case γ it is possible to define the set $IO(\gamma)$ of input/output sequences that may be observed using γ.

Definition 6 *Given $\gamma \in \mathcal{T}$, the input/output sequences that may be observed*

using γ are:

$$IO(\gamma) = \begin{cases} \{\epsilon\} & \text{if } \gamma = null \\ \bigcup_{y \in Y} \{\langle x/y \rangle\} IO(f(y)) & \text{if } \gamma = (x, f) \end{cases}$$

The first rule simply states that if the adaptive test case is *null* then no input/output behaviour is seen and thus the empty sequence is observed. The second rule is recursive, stating that given $y \in Y$, γ may lead to an input/output behaviour in the form of x/y followed by some input/output behaviour formed by applying $f(y)$. For example, the adaptive test case γ given in Figure 1 has: $IO(\gamma) = \{\langle a/0 \rangle, \langle a/1, b/0 \rangle, \langle a/1, b/1 \rangle\}$. Clearly, $IO(\gamma)$ is finite and every element of $IO(\gamma)$ is finite.

Proposition 1 *If $\gamma_1, \gamma_2 \in \Gamma$ and $IO(\gamma_1) = IO(\gamma_2)$ then $\gamma_1 = \gamma_2$.*

Definition 7 *The length of an adaptive test case γ is defined by:*

$$length(\gamma) = \begin{cases} 0 & \text{if } \gamma = null \\ 1 + max_{y \in Y} length(f(y)) & \text{if } \gamma = (x, f) \end{cases}$$

The length of $\gamma \in \mathcal{T}$ is the length of the longest input/output sequence that may result from applying γ to an implementation.

Given an adaptive test case γ, it is also possible to define the size $|\gamma|$ of γ to be the number of nodes in the tree that represents γ.

Definition 8 *Given $\gamma \in \mathcal{T}$, the* size $|\gamma|$ *of γ is defined by the following rules:*

$$\begin{aligned} |null| &= 1 \\ |(x, f)| &= 1 + \sum_{y \in Y} |f(y)| \end{aligned}$$

Consider the adaptive test case γ given in Figure 1. Here $|\gamma| = 1 + |null| + |(b, f')|$. Thus, since $|(b, f')| = 1 + |f'(0)| + |f'(1)| = 1 + |null| + |null| = 3$, $|\gamma| = 5$.

Throughout this paper Γ will denote a given (finite) set of adaptive test cases to be used in testing. Thus $\Gamma \subseteq \mathcal{T}$.

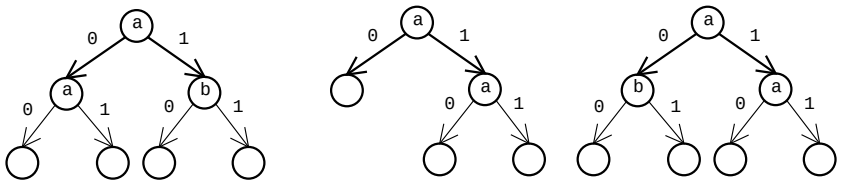

Fig. 2. Adaptive test cases γ_1, γ_2, and γ_3

3 Conditions for test case elimination

This section will explore some conditions under which, by applying one adaptive test case γ_1 before another adaptive test case γ_2, it may not be necessary to use γ_2. Consider the first two adaptive test cases γ_1 and γ_2 in Figure 2. Let us suppose that the use of γ_1 leads to the input/output sequence $\langle a/0, a/1\rangle$. Then, since the IUT is deterministic, we know that the response of the IUT to γ_2 will be $\langle a/0\rangle$. Thus there is no need to apply γ_2. Naturally, if the application of γ_1 had led to the input/output sequence $\langle a/1, b/0\rangle$ or to the input/output sequence $\langle a/1, b/1\rangle$ then the response of the IUT to γ_2 would not be determined and thus γ_2 should still be applied.

Let us now suppose that we intend to execute an arbitrary adaptive test case γ_2 after any other γ_1, with γ_1 being followed by a postamble that returns the system to its initial state. Then the response of the IUT to γ_1 might determine the response of the IUT to γ_2 if one of the possible responses to γ_2 is a prefix of some possible response to γ_1. This is captured below in Definition 10.

Given an input/output sequence $\sigma \in (X/Y)^*$ and an adaptive test case $\gamma \in \mathcal{T}$, the predicate $decides(\sigma, \gamma)$ will tell us if σ being observed in the IUT determines the response of the IUT to γ.

Definition 9 *The predicate* decides *is defined by the following rules:*

$$\begin{aligned} decides(\sigma, null) &= true \\ decides(\epsilon, (x, f)) &= false \\ decides(\langle x_1/y_1\rangle\sigma, (x, f)) &= (x_1 = x) \wedge decides(\sigma, f(y_1)) \end{aligned}$$

We clearly have the following.

Proposition 2 *Given $\sigma \in (X/Y)^*$ and $\gamma \in \mathcal{T}$, $decides(\sigma, \gamma)$ may be computed in $O(min(|\sigma|, length(\gamma)))$ time.*

Suppose we wish to apply the adaptive test cases from a given set $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ in the order $\langle \gamma_1, \gamma_2, \ldots, \gamma_n\rangle$. Then we could control the test execution using the following algorithm that prunes the set of adaptive test cases

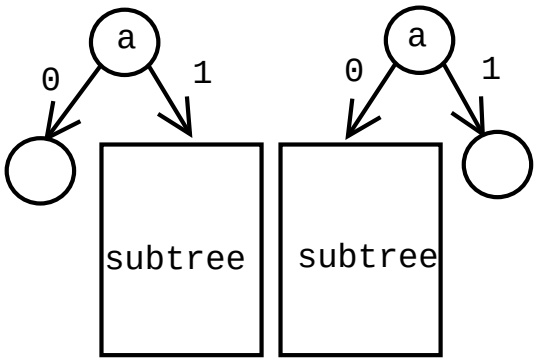

Fig. 3. Related adaptive test cases

yet to be executed on the basis of those input/output sequences that have already been observed.

Algorithm 1 *(1) Let* $\Delta = \langle \gamma_1, \gamma_2, \ldots, \gamma_n \rangle$
(2) while $\Delta \neq \epsilon$ *do*
(3) Let γ *denote the first element of* Δ.
(4) Test the IUT with γ *and let the observed input/output sequence be denoted* x/y.
(5) Remove γ *from* Δ.
(6) Remove from Δ *any element* γ' *such that* $decides(x/y, \gamma')$.
(7) od

Proposition 3 *The time taken in testing, using Algorithm 1, with the sequence* $\langle \gamma_1, \gamma_2, \ldots, \gamma_n \rangle$ *of elements of* $\mathcal{T}$ *is in* $O(n \sum_{i=1}^{n} length(\gamma_i))$.

Proof. First observe that at worst each adaptive test case is applied once and thus the time required to run the tests is in $O(\sum_{i=1}^{n} length(\gamma_i))$.

There are at most n iterations of the loop. Each iteration of the loop involves determining $decides(\sigma, \gamma)$ for some $\sigma \in (X/Y)^*$ and each remaining $\gamma \in \Delta$. Thus, the effort (in addition to executing a test) on each iteration of the loop is in $O(\sum_{i=1}^{n} length(\gamma_i))$. The result now follows. □

We will now explore the conditions under which one adaptive test case may determine the result of applying another.

Definition 10 *Let* $\gamma_1, \gamma_2 \in \mathcal{T}$. *We say that the response of the IUT to* γ_1 *may determine the response of the IUT to* γ_2 *if there is some* $\sigma_1 \in IO(\gamma_1)$ *and* $\sigma_2 \in IO(\gamma_2)$ *such that* $\sigma_2 \in pre(\sigma_1)$. *Whenever this is the case we write* $\gamma_2 \preceq \gamma_1$.

Note that $\preceq$ is not an ordering since it is not antisymmetric: there are distinct adaptive test cases γ_1 and γ_2 such that $\gamma_2 \preceq \gamma_1$ and $\gamma_1 \preceq \gamma_2$. Such a case is illustrated in Figure 3.

Proposition 4 *The relation $\preceq$ is reflexive but is not always symmetric and not always transitive.*

Proof. The fact that $\preceq$ is reflexive is an immediate consequence of the definition. Adaptive test cases γ_1 and γ_2 in Figure 2 demonstrate that $\preceq$ is not always symmetric.

In order to see that $\preceq$ is not always transitive consider the example in Figure 2. It is straightforward to check that $\gamma_2 \preceq \gamma_1$, $\gamma_3 \preceq \gamma_2$ but $\gamma_3 \not\preceq \gamma_1$. □

Clearly, if $\gamma_2 \preceq \gamma_1$ but $\gamma_1 \not\preceq \gamma_2$ it is desirable to use γ_1 before γ_2.

The predicate *sav*, defined below, may be used to decide whether $\gamma_1 \preceq \gamma_2$.

Definition 11 *The predicate* sav *is defined by the following rules:*

$$\begin{aligned} sav(\gamma, null) &= true \\ sav(null, (x, f)) &= false \\ sav((x_1, f_1), (x_2, f_2)) &= (x_1 = x_2) \wedge \exists y \in Y. sav(f_1(y), f_2(y)) \end{aligned}$$

The following result is clear.

Proposition 5 *Given $\gamma_1, \gamma_2 \in \mathcal{T}$, $\gamma_2 \preceq \gamma_1$ if and only if $sav(\gamma_1, \gamma_2)$.*

Proposition 6 *There is a constant c such that given $\gamma_1, \gamma_2 \in \mathcal{T}$, $sav(\gamma_1, \gamma_2)$ may be determined in time at most $c(|\gamma_1| + |\gamma_2|)$.*

Proof. First observe that the process of checking the right hand side of the first two rules requires constant time. The first part of the third rule also requires constant time. Let c be the sum of the corresponding three constants.

For all γ_1 and γ_2, let $g(\gamma_1, \gamma_2)$ denote the time taken to determine $sav(\gamma_1, \gamma_2)$. If $\gamma_1 = (x, f_1)$ and $\gamma_2 = (x, f_2)$ then $g(\gamma_1, \gamma_2) \leq c + \sum_{y \in Y} g(f_1(y), f_2(y))$.

Now the result follows by induction on $|\gamma_1| + |\gamma_2|$, proving that $g(\gamma_1, \gamma_2) \leq c(|\gamma_1| + |\gamma_2|)$. □

Based on the relation $\preceq$ we can define the following digraph.

Definition 12 *Given a set $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ of adaptive test cases, the* dependence digraph *is the digraph $G = (V, E)$ in which $V = \{v_1, \ldots, v_n\}$ and there is an edge $(v_i, v_j) \in E$ if and only if $\gamma_j \preceq \gamma_i$ and $\gamma_j \neq \gamma_i$.*

Since the dependence digraph may be constructed by determining $sav(\gamma_1, \gamma_2)$, for all $\gamma_1, \gamma_2 \in \Gamma$, the following is a consequence of Proposition 6.

Proposition 7 *Given a set Γ of adaptive test cases, its dependence digraph may be derived in time $O(n \sum_{\gamma \in \Gamma} |\gamma|)$.*

Proof. In order to find the dependence digraph it is sufficient to determine whether $\gamma_j \preceq \gamma_i$ for every ordered pair (γ_i, γ_j) of adaptive test cases in Γ with $\gamma_1 \neq \gamma_j$.

By Proposition 6, it is possible to decide whether $\gamma_j \preceq \gamma_i$ in $O(|\gamma_i|+|\gamma_j|)$. Thus, the dependence digraph may be constructed in $O(\sum_{\gamma_i, \gamma_j \in \Gamma . \gamma_i \neq \gamma_j} |\gamma_i| + |\gamma_j|)$. For each $\gamma \in \Gamma$ the term $|\gamma|$ appears $2n - 2$ times in the sum. Then the desired result follows. □

4 The complexity of the optimisation problem

In this section we prove that the problem of finding an ordering of a set of adaptive test cases, that minimizes the expected cost of testing, is NP-hard. We start by describing a classic NP-complete problem: the feedback arc set problem (see, for example, [2,7]).

Definition 13 *Given a digraph $G = (V, E)$ we say that a set $A \subseteq E$ is a* feedback arc set *of G if $(V, E \setminus A)$ is acyclic.*

Definition 14 *The* feedback arc set (FAS) *problem is: Given a digraph G, find a minimum size feedback arc set.*

The following result has been proved in [14].

Theorem 8 *The FAS problem is NP-complete.*

It is possible to prove that the problem of finding an optimal ordering of a set of adaptive test cases is NP-hard by reducing it to the problem above.

Theorem 9 *The problem of determining the optimal ordering of a set Γ of adaptive test cases is NP-hard.*

Proof. The proof will proceed by generating an instance of the problem from an instance of the FAS problem.

Suppose we have a digraph $G = (V, E)$ for which we wish to solve the FAS problem. Let n be the number of vertices of G. An ordering of the vertices of V may be represented by a one-to-one function f from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$.

Let $A = \{a_0\} \cup_{1 \leq i \leq n} \{a_i\}$ and $B = \{b_0\} \cup_{1 \leq i,j \leq n . i \neq j} \{b_{i,j}\}$. A will form the input alphabet and B will form the output alphabet.

We produce a set of adaptive test cases $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ in the following way.

Each element in Γ starts with the input a_0 and given $1 \leq i \leq n$, after $\langle a_0/b_0 \rangle$ γ_i gives an adaptive test case that has as input a_i combined with a function that maps any output to *null*. It is now sufficient to state the response of each γ_k after $\langle a_0/b_{i,j} \rangle$ for each $b_{i,j} \in B$. Given $1 \leq i, j \leq n$ with $i \neq j$ there are two cases to consider:

Case 1: $(v_i, v_j) \in E$. The adaptive test cases γ_i and γ_j both have input a_0 after $a_0/b_{i,j}$.

After $\langle a_0/b_{i,j}, a_0/b \rangle$ $(b \in B)$, the adaptive test case γ_i gives an adaptive test case with input a_0 combined with a function that maps any output to *null*.

After $\langle a_0/b_{i,j}, a_0/b \rangle$ $(b \in B)$ the adaptive test case γ_j becomes *null*.

Each γ_k with $1 \leq k \leq n$ and $k \neq i$, $k \neq j$, has a unique input a_k after $\langle a_0/b_{i,j} \rangle$. Any output is mapped to *null* after a_k.

Case 2: $(v_i, v_j) \notin E$. Each γ_k $(1 \leq k \leq n)$ has a unique input a_k, which leads to any output being mapped to *null*, after $\langle a_0/b_{i,j} \rangle$.

Now consider the conditions under which we have $\gamma_j \preceq \gamma_i$. Clearly this cannot occur due to any behaviour that starts with $\langle a_0/b_0 \rangle$. Suppose we have that $\gamma_j \preceq \gamma_i$ due to some behaviour starting with $\langle a_0/b \rangle$ for some $b \in B \setminus \{b_0\}$. Then this must be followed by the same input for γ_j and γ_i and thus this can only occur if $b = b_{i,j}$ or $b = b_{j,i}$. Further, since $\gamma_j \preceq \gamma_i$ we must have that $b = b_{i,j}$ and thus $(v_i, v_j) \in E$. If $(v_i, v_j) \in E$ then we have $\gamma_j \preceq \gamma_i$. Thus, $\gamma_j \preceq \gamma_i$ if and only if the edge $(v_i, v_j) \in E$ and so the dependence digraph for Γ is isomorphic to G.

Since the IUT is deterministic, there is only one possible $b \in B$ such that the IUT responds to a_0 with b when in its initial state. Thus the possible savings, represented by edges in the dependence digraph of Γ, are mutually exclusive: when the elements of Γ are applied at most one of these savings may occur. From this it is clear that the optimal ordering f of the elements of Γ is the ordering in which fewest edges of the dependence digraph for Γ go in the opposite direction to the ordering. Thus, since the dependence digraph of Γ is isomorphic to G, f is an ordering of the vertices of G that minimises the number of edges $(v_i, v_j) \in E$ such that $f(i) > f(j)$.

Observe that an ordering g defines the following arc feedback set: $\{(v_i, v_j) \in E | g(i) > g(j)\}$. Thus, since the set $\{(v_i, v_j) \in E | f(i) > f(j)\}$ is a minimum size such set it is a solution to the FAS problem for G. Further, the procedure described above creates an instance of the optimisation problem from an instance of the FAS problem in polynomial time. The result thus follows. □

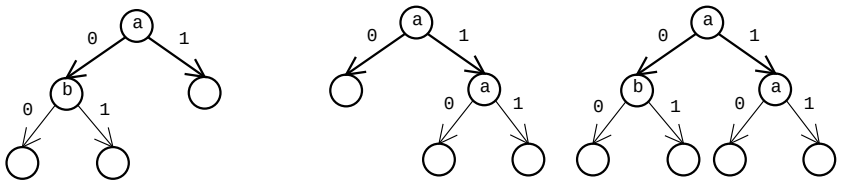

Fig. 4. Merging γ_1 and γ_2 to form γ_3

5 Simplifying the optimisation problem

Since the problem of finding an optimal ordering of a set of adaptive test cases is NP-hard it could be useful to reduce the number of adaptive test cases we must consider. This section describes two ways of reducing the scale of the optimisation problem and proves that these approaches do not conflict: by applying one approach we do not reduce the scope for applying the other.

5.1 Merging adaptive test cases

Given two adaptive test cases γ_1 and γ_2, it may be possible to combine these adaptive test cases to form one adaptive test case γ_3. The adaptive test case γ_3 should have the property that the information obtained by testing a deterministic IUT with γ_3 is identical to the information obtained by separately testing the IUT with γ_1 and γ_2. For example, Figure 4 has three adaptive test cases with the property that the first two may be merged to form the third. By applying γ_3 to a deterministic IUT we get the same information as if we had separately applied γ_1 and γ_2 to the IUT since each possible response to γ_3 is also a possible response to one of γ_1 and γ_2, and conversely.

When two adaptive test cases are merged, the size of the optimisation problem is reduced. Next we consider some conditions under which this may be done and we define an algorithm that merges adaptive test cases.

Definition 15 $\gamma_3 \in \mathcal{T}$ *is the result of* merging $\gamma_1, \gamma_2 \in \mathcal{T}$ *if and only if* $Pre(IO(\gamma_1)) \cup Pre(IO(\gamma_2)) = Pre(IO(\gamma_3))$.

Observe that, by Proposition 1, if we can merge two adaptive test cases γ_1 and γ_2 the result is unique.

Definition 16 $\gamma_1, \gamma_2 \in \mathcal{T}$ *are said to be* compatible *if and only if there is no pair of sequences* $\langle x_1/y_1, x_2/y_2, \ldots, x_{k-1}/y_{k-1}, x_k/y_k \rangle \in Pre(IO(\gamma_1))$ *and* $\langle x_1/y_1, x_2/y_2, \ldots, x_{k-1}/y_{k-1}, x'_k/y'_k \rangle \in Pre(IO(\gamma_2))$ *with* $x_k \neq x'_k$ $(k \geq 1)$.

Thus, γ_1 and γ_2 are compatible if there exists no input/output sequence σ that might result from both γ_1 and γ_2 such that γ_1 and γ_2 apply different input

values after σ. It is clear that in order to be able to merge two adaptive test cases γ_1 and γ_2 it is necessary that they are compatible: otherwise the resultant adaptive test case γ_3 should provide two different input values after some input/output sequence σ, which is not allowed by the definition of adaptive test cases.

Definition 17 *The function* compatible *decides whether two adaptive test cases are compatible and is defined by:*

$$\begin{aligned} compatible(\gamma, null) &= true \\ compatible(null, \gamma) &= true \\ compatible((x_1, f_1), (x_2, f_2)) &= (x_1 = x_2) \wedge \forall y \in Y.compatible(f_1(y), f_2(y)) \end{aligned}$$

Proposition 10 *$\gamma_1, \gamma_2 \in \mathcal{T}$ are compatible if and only if $compatible(\gamma_1, \gamma_2) = true$.*

The proof of the following result is similar to that of Proposition 6.

Proposition 11 *There exists a constant c_c such that, given $\gamma_1, \gamma_2 \in \mathcal{T}$, $compatible(\gamma_1, \gamma_2)$ may be computed in time at most $c_c(|\gamma_1| + |\gamma_2|)$.*

Definition 18 *Given two compatible adaptive test cases γ_1 and γ_2, the result of* merging *γ_1 and γ_2 is defined by:*

$$\begin{aligned} merge(\gamma, null) &= \gamma \\ merge(null, \gamma) &= \gamma \\ merge((x, f_1), (x, f_2)) &= (x, \{y \rightarrow merge(f_1(y), f_2(y)) | y \in Y\}) \end{aligned}$$

The proof of the following is similar to that for Proposition 6.

Proposition 12 *There exists a constant c_m such that, given $\gamma_1, \gamma_2 \in \mathcal{T}$ that may be merged, $merge(\gamma_1, \gamma_2)$ may be found in time $c_m(|\gamma_1| + |\gamma_2|)$.*

We now explore links between the function *merge*, the definition of two adaptive test cases being compatible, and the definition of one adaptive test case γ_3 being the result of merging two adaptive test cases γ_1 and γ_2.

Proposition 13 *If $\gamma_1, \gamma_2 \in \mathcal{T}$ are compatible then $merge(\gamma_1, \gamma_2)$ is the result of merging γ_1 and γ_2.*

Proof. First observe that, by definition, if γ_1 and γ_2 are compatible then $merge(\gamma_1, \gamma_2)$ is defined. Let $\gamma_3 = merge(\gamma_1, \gamma_2)$.

We prove the result by induction on the length of γ_3. The base case, where

$\gamma_3 = null$, is clear. Inductive hypothesis: if $\gamma_3 = merge(\gamma_1, \gamma_2)$ has length less than k then $Pre(IO(\gamma_1)) \cup Pre(IO(\gamma_2)) = Pre(IO(\gamma_3))$. Now suppose that $length(\gamma_3) = k$.

The result clearly holds if $\gamma_1 = null$ (as $\gamma_2 = \gamma_3$) or $\gamma_2 = null$ (as $\gamma_1 = \gamma_3$). We may thus assume that there exists $x \in X$, f_1, f_2, f_3 such that $\gamma_1 = (x, f_1)$, $\gamma_2 = (x, f_2)$, and $\gamma_3 = (x, f_3)$. Further, for all $y \in Y$, $f_1(y)$ and $f_2(y)$ are compatible.

It is sufficient to prove that $Pre(IO(\gamma_1)) \cup Pre(IO(\gamma_2)) = Pre(IO(\gamma_3))$. We will first prove by contradiction that $Pre(IO(\gamma_1)) \cup Pre(IO(\gamma_2)) \subseteq Pre(IO(\gamma_3))$. Suppose there is some $\langle x/y \rangle \sigma \in Pre(IO(\gamma_1)) \cup Pre(IO(\gamma_2)) \setminus Pre(IO(\gamma_3))$ $(x \in X, y \in Y)$, then $\sigma \in Pre(IO(f_1(y))) \cup Pre(IO(f_2(y))) \setminus Pre(IO(f_3(y)))$. By the definition of *merge*, $f_3(y) = merge(f_1(y), f_2(y))$. By the inductive hypothesis, since $length(f_3(y)) < length(\gamma_3)$, $merge(f_1(y), f_2(y))$ is the result of merging $f_1(y)$ and $f_2(y)$ and so $Pre(IO(f_1(y))) \cup Pre(IO(f_2(y))) = Pre(IO(f_3(y)))$. This provides a contradiction as required.

We will now prove that $Pre(IO(\gamma_3)) \subseteq (Pre(IO(\gamma_1)) \cup Pre(IO(\gamma_2)))$, also by contradiction. Let us suppose that there is some $\langle x/y \rangle \sigma \in Pre(IO(\gamma_3)) \setminus (Pre(IO(\gamma_1)) \cup Pre(IO(\gamma_2)))$ $(x \in X, y \in Y)$. Then $\sigma \in Pre(IO(f_3(y))) \setminus (Pre(IO(f_1(y))) \cup Pre(IO(f_2(y))))$. By definition of *merge*, $f_3(y) = merge(f_1(y), f_2(y))$. By the inductive hypothesis, since $length(f_3(y)) < length(\gamma_3)$, $merge(f_1(y), f_2(y))$ is the result of merging $f_1(y)$ and $f_2(y)$ and so $Pre(IO(f_3(y))) = Pre(IO(f_1(y))) \cup Pre(IO(f_2(y)))$. This provides a contradiction as required. □

Proposition 14 *It is possible to merge adaptive test cases if and only if they are compatible.*

Proof. By Proposition 13, it is possible to merge γ_1 and γ_2 if they are compatible. It thus suffices to prove that if γ_1 and γ_2 can be merged then they are compatible. Proof by contradiction: suppose that γ_1 and γ_2 can be merged but they are not compatible. Let γ_3 denote the result of merging γ_1 and γ_2.

Since γ_1 and γ_2 are not compatible there exists $\sigma = \langle x_1/y_1, \ldots, x_{m-1}/y_{m-1} \rangle$, $\sigma\langle x_m/y_m \rangle \in Pre(IO(\gamma_1))$, and $\sigma\langle x'_m/y'_m \rangle \in Pre(IO(\gamma_2))$ with $x_m \neq x'_m$. Now observe that, by definition, $\sigma\langle x_m/y_m \rangle \in Pre(IO(\gamma_3))$ and $\sigma\langle x'_m/y'_m \rangle \in Pre(IO(\gamma_3))$. This contradicts the definition of an adaptive test case, since γ_3 must allow two different input values after the input/output sequence σ. The result thus follows. □

The following results are immediate consequences of the definitions.

Proposition 15 *If $\gamma_1, \gamma_2 \in \mathcal{T}$ are compatible then $\gamma_1 \preceq merge(\gamma_1, \gamma_2)$ and $\gamma_2 \preceq merge(\gamma_1, \gamma_2)$.*

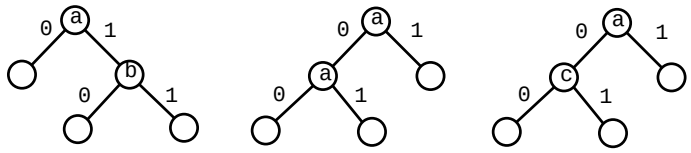

Fig. 5. Adaptive test cases γ_1, γ_2, and γ_3

Proposition 16 *If $\gamma_1, \gamma_2 \in \mathcal{T}$ are compatible then $|merge(\gamma_1, \gamma_2)| \leq |\gamma_1| + |\gamma_2|$.*

By merging compatible adaptive test cases it is possible to reduce the number of adaptive test cases considered.

Definition 19 *We say that a set Γ of adaptive test cases is* irreducible *if no two elements of Γ are compatible. Otherwise we say that it is* reducible.

It will be assumed that any set of adaptive test cases considered is irreducible: where there are compatible adaptive test cases these are merged before we determine the order in which the adaptive test cases will be applied in testing.

Observe that given a reducible set Γ of adaptive test cases, there may be more than one way in which to merge the elements of Γ in order to produce an irreducible set. To see this, consider the adaptive test cases in Figure 5. Here we may merge γ_1 and γ_2 or γ_1 and γ_3. In each case we are left with two adaptive test cases that cannot be merged.

Algorithm 2 takes a sequence of n adaptive test cases and returns an irreducible set. In this algorithm, $remove(\Gamma, \gamma_j)$ denotes the set formed from Γ by removing the element γ_j.

Algorithm 2 *(1) Input Γ;*
(2) i=1;
(3) while ($i < |\Gamma|$) do
(4) j=i+1;
(5) while ($j \leq |\Gamma|$) do
(6) if compatible(γ_i, γ_j) then
(a) $\gamma_i = merge(\gamma_i, \gamma_j)$;
(b) $\Gamma = remove(\Gamma, \gamma_j)$;
(c) For all k with $j < k \leq |\Gamma|$, relabel γ_k as γ_{k-1};
(7) else j=j+1;
(8) od
(9) i=i+1;
(10) od
(11) Output Γ;

Let us suppose that we are given adaptive test cases $\gamma_1 = (x, f_1)$ and $\gamma_2 = (x, f_2)$ that are not compatible but that $f_1(y)$ and $f_2(y)$ are compatible for all

$y \in Y$ except for one output $y_i \in Y$. When we determine $compatible(\gamma_1, \gamma_2)$ we first calculate $compatible(f_1(y_1), f_2(y_1))$ for some $y_1 \in Y$ and if this is $true$ ($i \neq 1$) then we next calculate $compatible(f_1(y_2), f_2(y_2))$ for some $y_2 \in Y \backslash \{y_1\}$. This process continues until we calculate $compatible(f_1(y_i), f_2(y_i))$. Thus, the cost of determining $compatible(\gamma_1, \gamma_2)$ depends upon the order in which we consider the elements of Y. In order to reason about the relative costs of applying *compatible* to different pairs of adaptive test cases we assume that the order in which the individual values for $compatible(f_1(y), f_2(y))$ ($y \in Y$) are calculated is determined by some fixed ordering $\langle y_1, \ldots, y_m \rangle$ of the elements of Y.

We will now explore the time complexity of Algorithm 2.

Lemma 17 *If $\gamma_1, \gamma_2 \in \mathcal{T}$ are compatible and $\gamma_3 = merge(\gamma_1, \gamma_2)$, the cost of determining compatible(γ, γ_3) is bounded above by the sum of the costs of determining compatible(γ, γ_1) and compatible(γ, γ_2).*

Proof. By induction on $|\gamma_1| + |\gamma_2|$. The base case, when $\gamma_1 = null$ and $\gamma_2 = null$, follows immediately. Similarly, the result is immediate if either $\gamma_1 = null$ or $\gamma_2 = null$.

Inductive hypothesis: the result holds if $|\gamma_1| + |\gamma_2| < k$ ($k > 2$). Suppose $|\gamma_1| + |\gamma_2| = k$.

Let us consider the case where $\gamma_1 = (x, f_1)$, $\gamma_2 = (x, f_2)$, $\gamma_3 = (x, f_3)$, and $\gamma = (x_1, f)$. The result clearly holds if $x \neq x_1$ so assume $x = x_1$. There are now two cases.

Case 1: $compatible(\gamma, \gamma_3) = true$. Then the cost of determining $compatible(\gamma, \gamma_i)$ ($1 \leq i \leq 3$) is a constant plus the sum of the costs of determining $compatible(f(y), f_i(y))$ for all $y \in Y$. The result now follows from the inductive hypothesis.

Case 2: $compatible(\gamma, \gamma_3) = false$. Let $\langle x_1/y_1, \ldots, x_m/y_m, x/y \rangle \in Pre(IO(\gamma))$ and $\langle x_1/y_1, \ldots, x_m/y_m, x'/y' \rangle \in Pre(IO(\gamma_3))$ be the elements first found that show that $compatible(\gamma, \gamma_3) = false$. Then the cost of determining that $compatible(\gamma, \gamma_3) = false$ is the sum of:

(1) the sum over the $y \in Y$ that are considered before y_1 of the cost of determining that $compatible(f(y), f_3(y)) = true$ plus
(2) the cost of determining that $compatible(f(y_1), f_3(y_1)) = false$.

Observe that for all $y \in Y$ that are considered before y_1 we must have that $compatible(f(y_1), f_i(y_1)) = true$ ($1 \leq i \leq 2$). Thus, the cost of determining $compatible(\gamma, \gamma_i)$ ($1 \leq i \leq 2$) is at least:

(1) the sum over the $y \in Y$ that are considered before y_1 of the cost of

determining that $compatible(f(y), f_i(y)) = true$ plus
(2) the cost of determining the value of $compatible(f(y_1), f_i(y_1))$.

The result now follows by applying the inductive hypothesis. □

Proposition 18 *If the set Γ' of adaptive test cases is formed from the set Γ of adaptive test cases through a sequence of applications of merge, then for any $\gamma \in \mathcal{T}$ the total cost of determining compatible(γ'_i, γ) for all $\gamma'_i \in \Gamma'$ is at most the total cost of determining compatible(γ_i, γ) for all $\gamma_i \in \Gamma$.*

Proof. This follows from Lemma 17 and a simple proof by induction on the number of applications of *merge*. □

Proposition 19 *Given a set Γ of n adaptive test cases, Algorithm 2 requires time in $O(n \sum_{\gamma \in \Gamma} |\gamma|)$.*

Proof. From Propositions 11 and 12 we know that there exist $c_m, c_c > 0$ such that the cost of determining $merge(\gamma_1, \gamma_2)$ is bounded above by $c_m(|\gamma_1| + |\gamma_2|)$ and the cost of determining $compatible(\gamma_1, \gamma_2)$ is bounded above by $c_c(|\gamma_1| + |\gamma_2|)$.

We can partition the set Γ into subsets Γ_M and Γ_U such that Γ_U contains the adaptive test cases from Γ that are not merged with other adaptive test cases when Algorithm 2 is applied. Thus, the application of Algorithm 2 involves the merging of the elements in Γ_M. We will prove that $g(\Gamma, \Gamma_M) = c_c(n \sum_{\gamma \in \Gamma} |\gamma|) + c_m(|\Gamma_M| \sum_{\gamma \in \Gamma_M} |\gamma|)$ is an upper bound of the execution time of Algorithm 2. We will apply induction on the number of uses of *merge* in the application of the algorithm. The base case, where no elements of Γ are merged, follows immediately from Proposition 11.

Inductive hypothesis: if the application of Algorithm 2 involves less than k uses of *merge* ($k > 0$) then Algorithm 2 requires time at most $g(\Gamma, \Gamma_M)$.

Suppose the application of Algorithm 2 requires k uses of *merge* ($k > 0$). It is now sufficient to prove that Algorithm 2 requires time at most $g(\Gamma, \Gamma_M)$.

Suppose that the last pair to be merged in the application of Algorithm 2 is (γ'_i, γ_j) $(i < j)$ and $\gamma' = merge(\gamma'_i, \gamma_j)$.

Let $\Gamma' = \{\gamma'_1, \ldots, \gamma'_m\}$ denote the set of adaptive test cases, formed from Γ, immediately before γ'_i and γ_j are merged. We may now compare the time taken in applying Algorithm 2 to the two sets Γ and $\Gamma \setminus \{\gamma_j\}$. There are two possible effects, on the execution time of Algorithm 2, of the addition of γ_j to $\Gamma \setminus \{\gamma_j\}$ to form Γ:

(1) The effect on the cost of calculating $compatible(\gamma'_i, \gamma_j)$ for all $\gamma'_i \in \Gamma' \setminus$

$\{\gamma_j\}$. By Proposition 18 this is bounded above by the cost of determining *compatible*(γ_i, γ_j) for all $\gamma_i \in \Gamma \setminus \{\gamma_j\}$ which in turn is bounded above by $c_c \sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}(|\gamma|+|\gamma_j|)$. This is equal to $c_c((\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|)+(\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma_j|)$ which may be simplified to $c_c((\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|)+(n-1)|\gamma_j|)$.

(2) The cost of merging γ_i' and γ_j. This is at most $c_m(|\gamma_i'|+|\gamma_j|)$ which is less than or equal to $c_m \sum_{\gamma\in\Gamma_M}|\gamma|$ since, by Proposition 16, $|\gamma_i'| \leq \sum_{\gamma\in\Gamma_M\setminus\{\gamma_j\}}|\gamma|$.

Thus, by the inductive hypothesis, the cost of applying the algorithm to Γ is at most:

$$g(\Gamma \setminus \{\gamma_j\}, \Gamma_M \setminus \{\gamma_j\}) + c_c((\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|)+(n-1)|\gamma_j|) + c_m \sum_{\gamma\in\Gamma_M}|\gamma|$$

But

$$g(\Gamma \setminus \{\gamma_j\}, \Gamma_M \setminus \{\gamma_j\}) = c_c((n-1)\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|) + c_m(|\Gamma_M \setminus \{\gamma_j\}|\sum_{\gamma\in\Gamma_M\setminus\{\gamma_j\}}|\gamma|)$$

Thus the cost of applying the algorithm to Γ is at most

$$c_c((n-1)\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|) + c_m(|\Gamma_M \setminus \{\gamma_j\}|\sum_{\gamma\in\Gamma_M\setminus\{\gamma_j\}}|\gamma|) + c_c((\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|)+(n-1)|\gamma_j|) + c_m\sum_{\gamma\in\Gamma_M}|\gamma|$$

which may be simplified to

$$c_c(n\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|+(n-1)|\gamma_j|) + c_m((|\Gamma_M|-1)\sum_{\gamma\in\Gamma_M\setminus\{\gamma_j\}}|\gamma| + \sum_{\gamma\in\Gamma_M}|\gamma|)$$

Further, $c_m((|\Gamma_M|-1)|\sum_{\gamma\in\Gamma_M\setminus\{\gamma_j\}}|\gamma| + \sum_{\gamma\in\Gamma_M}|\gamma|)$ is bounded above by the term $c_m(|\Gamma_M|\sum_{\gamma\in\Gamma_M}|\gamma|)$. Thus, the upper bound may be replaced by the following.

$$c_c(n\sum_{\gamma\in\Gamma\setminus\{\gamma_j\}}|\gamma|+(n-1)|\gamma_j|) + c_m(|\Gamma_M|\sum_{\gamma\in\Gamma_M}|\gamma|)$$

The result now follows from observing that this is bounded above by $g(\Gamma, \Gamma_M) = c_c(n\sum_{\gamma\in\Gamma}|\gamma|) + c_m(|\Gamma_M|\sum_{\gamma\in\Gamma_M}|\gamma|)$. □

We have seen that the order of the application of adaptive test cases could be important. However, there may be adaptive test cases in the test suite used whose relative order is irrelevant. When this is identified, the problem of determining the optimal ordering may be reduced to that of determining the optimal ordering amongst the elements within a subset of adaptive test cases. This breaks up the overall optimisation problem into a set of simpler problems.

Suppose that we are considering two adaptive test cases γ_1 and γ_2 with $\gamma_1 \not\preceq \gamma_2$ and $\gamma_2 \not\preceq \gamma_1$ so that it appears that the relative order of γ_1 and γ_2 is irrelevant. However, since $\preceq$ is not transitive, it is possible that there is some γ_3 such that $\gamma_1 \preceq \gamma_3$ and $\gamma_3 \preceq \gamma_2$. Thus, in order to define some notion of independence we generate an equivalence relation from $\preceq$.

Definition 20 *The relation $\sim$ is the transitive closure of the union of $\preceq$ and its inverse relation. If $\gamma_1 \sim \gamma_2$ then we say that γ_1 and γ_2 are* dependent. *Otherwise we say that they are* independent *and write $\gamma_1 \not\sim \gamma_2$.*

Observe that $\sim$ and $\not\sim$ have an implicit parameter: the set Γ. In Lemma 22 we will prove that if two adaptive test cases are compatible then they are dependent. The relation $\sim$ is an equivalence relation and so we may consider its equivalence classes. Given two equivalence classes Q_1 and Q_2 if $Q_1 \neq Q_2$, $\gamma_1 \in Q_1$, and $\gamma_2 \in Q_2$, γ_1 and γ_2 must be independent. It is then possible to split the set of adaptive test cases used into these equivalence classes and determine an order of application for the elements of each equivalence class. This observation may simplify the problem of finding an optimal ordering.

Naturally, the equivalence classes are simply the weakly connected components of the dependence digraph. Thus, since the weakly connected components of $G = (V, E)$ may be found in $O(|V|+|E|)$ [19], the following result is immediate.

Proposition 20 *The set of equivalence classes of $\sim$ may be computed in a time in $O(|\Gamma|^2)$.*

Proof. We just have to observe that the number of vertices is $|\Gamma|$ so that the number of edges is bounded above by $|\Gamma|^2$. □

We have seen two different approaches that simplify the optimization problem: dividing the set of adaptive test cases into a set of equivalence classes and merging adaptive test cases where possible. In Theorem 24 we prove that these approaches do not conflict: by applying one approach we do not reduce the scope for applying the other.

Lemma 21 *Given $\gamma, \gamma_1, \gamma_2 \in \mathcal{T}$, if $\gamma = merge(\gamma_1, \gamma_2)$ and $\sigma \in IO(\gamma)$ then we have either $\sigma \in IO(\gamma_1)$ or $\sigma \in IO(\gamma_2)$.*

Proof. This follows from Proposition 13 and Definition 18. □

Lemma 22 *If we can merge γ_1 and γ_2 then $\gamma_1 \sim \gamma_2$.*

Proof. Suppose that we can merge γ_1 and γ_2 and let $\gamma = merge(\gamma_1, \gamma_2)$.

Let $\sigma = \langle x_1/y_1, \ldots, x_m/y_m \rangle \in IO(\gamma)$. By Lemma 21, $\sigma \in IO(\gamma_1) \cup IO(\gamma_2)$. Without loss of generality, $\sigma \in IO(\gamma_1)$. If $\sigma \in Pre(IO(\gamma_2))$ then $\gamma_1 \preceq \gamma_2$, and so $\gamma_1 \sim \gamma_2$, as required. Let us now assume that $\sigma \notin Pre(IO(\gamma_2))$.

Let σ' denote the longest prefix of σ contained in $Pre(IO(\gamma_2))$. Thus $\sigma' = \langle x_1/y_1, \ldots, x_k/y_k \rangle$ for some $k < m$. Suppose there is some extension, $\sigma' \langle x/y \rangle$ of σ' in $Pre(IO(\gamma_2))$. Observe that, since γ_1 and γ_2 may be merged, by Proposition 14, γ_1 and γ_2 are compatible. Thus $x = x_{k+1}$, contradicting the maximality of σ'. Thus $\sigma' \in IO(\gamma_2)$ and so $\gamma_2 \sim \gamma_1$ as required. □

Lemma 23 *Let $\gamma, \gamma_1, \gamma_2 \in \mathcal{T}$ with $\gamma_1 \not\preceq \gamma$, $\gamma \not\preceq \gamma_1$, $\gamma_2 \not\preceq \gamma$, $\gamma \not\preceq \gamma_2$, and $\gamma_3 = merge(\gamma_1, \gamma_2)$, then $\gamma \not\preceq \gamma_3$ and $\gamma_3 \not\preceq \gamma$.*

Proof. We will prove the result by contradiction. There are two cases to consider.

Case 1: $\gamma_3 \preceq \gamma$. Thus there exists two input/output sequences $\sigma \in IO(\gamma_3)$ and $\sigma' \in IO(\gamma)$ such that $\sigma \in pre(\sigma')$. By Lemma 21, $\sigma \in IO(\gamma_1) \cup IO(\gamma_2)$. Without loss of generality, we can take $\sigma \in IO(\gamma_1)$. Thus, $\gamma_1 \preceq \gamma$, providing a contradiction as required.

Case 2: $\gamma \preceq \gamma_3$. Thus there exist two input/output sequences $\sigma \in IO(\gamma)$ and $\sigma' \in IO(\gamma_3)$ such that $\sigma \in pre(\sigma')$. By Lemma 21, $\sigma' \in IO(\gamma_1) \cup IO(\gamma_2)$. Without loss of generality, we can take $\sigma' \in IO(\gamma_1)$. Thus, $\gamma \preceq \gamma_1$, providing a contradiction as required. □

The following result shows that we cannot combine two equivalence classes of $\sim$ by merging two adaptive test cases.

Theorem 24 *Let γ_i and γ_j be two compatible adaptive test cases from Γ, $\gamma = merge(\gamma_i, \gamma_j)$, and $\Gamma' = \Gamma \setminus \{\gamma_1, \gamma_2\} \cup \{\gamma\}$. For every equivalence class Q of Γ under $\sim$ let Q' denote the corresponding set of adaptive test cases from Γ': if $\gamma_i \in Q$ or $\gamma_j \in Q$ then $Q' = Q \setminus \{\gamma_1, \gamma_2\} \cup \{\gamma\}$ and otherwise $Q' = Q$. Then for any two distinct equivalence class Q_1 and Q_2 of Γ under $\sim$, no element of Q'_1 is related to an element of Q'_2 under $\sim$.*

Proof. By Lemma 22, γ_i and γ_j are in the same equivalence class of Γ under $\sim$. There are two cases to consider.

Case 1: $\gamma_i, \gamma_j \in Q_1 \cup Q_2$. Without loss of generality, $\gamma_i, \gamma_j \in Q_1$. Since Q_1 is an equivalence relation of Γ under $\sim$, no element of $Q'_1 \setminus \{\gamma\}$ is related to an element of Q'_2 under $\sim$. Further, by Lemma 23, γ is not related to any element of Q'_2 under $\sim$.

Case 2: $\gamma_i, \gamma_j \notin Q_1 \cup Q_2$. Let Q_k denote the equivalence class of Γ that contains γ_i and γ_j. By Lemma 23 no element of $\Gamma' \setminus Q'_k$ is related to any element of Q'_k under $\sim$ and so the result follows from $Q_1 = Q'_1$ and $Q_2 = Q'_2$ being equivalence classes of Γ under $\sim$. □

6 Ordering based on the dependence digraph

In this section we will study the problem of finding an ordering based on the dependence digraph. First we consider the special case where the dependence digraph G is acyclic. It transpires that in this case an exact solution may be found in polynomial time.

The following is a useful property of acyclic digraphs.

Proposition 25 *If digraph $G = (V, E)$ is acyclic and $|V| > 0$ then there exists some $v \in V$ such that $indegree_E(v) = 0$.*

Thus, given an acyclic digraph G that represents the relation $\preceq$ for some set Γ of adaptive test cases, it is possible to choose some $\gamma \in \Gamma$ such that the application of another element of Γ cannot lead to γ not being required. Thus, in choosing an order in which to apply the elements of Γ it is acceptable to start with γ.

Proposition 26 *Let us suppose that the digraph $G = (V, E)$ is acyclic and $v \in V$. Then the digraph $G \setminus \{v\}$ is acyclic.*

From this it is clear that if the dependence digraph G is acyclic, then having chosen some $\gamma \in \Gamma$ as described above, when γ is removed from Γ the resultant digraph is acyclic.

Based on these results, we get the following algorithm for the case when $\preceq$ defines an acyclic dependency digraph. This algorithm essentially chooses some ordering based on a directed acyclic graph (DAG).

Algorithm 3 *(1) Input the dependence digraph $G = (V, E)$, where $v_i \in V$ represents adaptive test case γ_i.*
(2) Set $T = \epsilon$, $G_0 = (V_0, E_0) = (V, E)$, $k = 0$.
(3) While ($V_k \neq \emptyset$) do
(4) Choose some $v_i \in V_k$ such that $indegree_{E_k}(v_i) = 0$.

(5) Set $T = T\langle\gamma_i\rangle$, $k = k + 1$, $G_k = (V_k, E_k) = G_{k-1} \setminus \{v_i\}$.
(6) od
(7) Output T

The following result is clear.

Proposition 27 *Algorithm 3 has computational complexity* $O(|\Gamma|^2)$.

Thus, when the dependency digraph is acyclic the problem of finding an optimal ordering for the adaptive test cases may be solved in polynomial time.

We will now give an algorithm, for the general case where G contains cycles. This algorithm is also based on the dependence digraph.

Algorithm 4 *(1) Input the dependence digraph* $G = (V, E)$, *where* $v_i \in V$ *represents adaptive test case* γ_i.
(2) Find a feedback arc set A.
(3) Set $G' = (V, E \setminus A)$.
(4) Return the result of applying Algorithm 3 to G'.

Note that at step 2, ideally A is the solution to the feedback arc set problem. However, since this problem is NP-complete, typically we will apply some heuristic in order to generate a possibly large feedback arc set A. A number of low-order polynomial time heuristics have been developed (see for example [4,5]). See [6] for an overview of the feedback arc set problem and some of the heuristics that have been developed for this problem. It is interesting to note that this problem is solvable in polynomial time for undirected graphs and planar graphs [16].

7 Conclusion and discussion

Adaptive test cases are used when testing a state-based system. Where the implementation under test (IUT) is known to be deterministic, the response of the IUT to one adaptive test case γ_1 may fully determine the response of the IUT to another adaptive test case γ_2. We have introduced a test application algorithm that utilises this: when the response of the IUT to an adaptive test case γ is determined by previous tests, γ is not used in testing. It transpires that, when this algorithms is used, the order in which the adaptive test cases are applied may affect the expected cost of testing. This paper has considered the problem of finding an ordering that minimises the expected cost of testing.

The relation, which states when the use of one adaptive test case may lead to another not being used, has been represented as the dependence digraph. This paper has shown that while the overall optimisation problem is NP-hard,

when the dependence digraph is acyclic the problem may be solved in low order polynomial time.

By considering the dependence digraph we have applied an abstraction, in which it is deemed to be sufficient to consider only the relation $\preceq$. When the dependence digraph is acyclic this abstraction leads to no loss of precision. However, in general this might throw away some useful information, such as the expected effort saved by utilising an ordering $\gamma_1 \preceq \gamma_2$ or the probability that the response to γ_1 actually determines the response to γ_2. One natural way of introducing such information is to weight the edges of the dependence digraph. However, there may also be dependencies between relations of $\preceq$. For example, the saving due to having γ_{i_2} before γ_{i_1} may preclude the saving due to having γ_{j_2} before γ_{j_1} and so we lose information if we simply apply weights to the edges of the dependence digraph. Future work will consider how additional information may be used when determining the optimal order of application of a set of adaptive test cases.

We have proved that the optimisation problem is NP-hard and described two ways of reducing the size of the optimisation problem: by merging adaptive test cases and by splitting the set of adaptive test cases into a set of equivalence classes that may be considered separately. We have proved that these two approaches are practical (they take low-order polynomial time) and they do not affect one another: merging adaptive test cases cannot lead to equivalence classes being combined.

This paper has focussed on the case where the IUT is known to be deterministic and thus will always respond to an input sequence with the same output sequence. When the IUT is non-deterministic, it is sometimes possible to make a fairness assumption: it is assumed that there is some k such that the use of an adaptive test case k times will lead to all possible responses being observed. When such an assumption is made, and each adaptive test case is applied k times, it is possible for the responses of the IUT to one adaptive test case γ_1 to be capable of determining the response to another adaptive test case γ_2. An alternative might be to apply a probabilistic argument based on repeated applications of an adaptive test case. Future work will consider the problem of finding an optimal ordering of a set of adaptive test cases when the IUT is non-deterministic.

Acknowledgements

This work was supported in part by Leverhulme Trust grant number F/00275/D, Testing State Based Systems, Natural Sciences and Engineering Research Council (NSERC) of Canada under grant number RGPIN976, and Engineer-

ing and Physical Sciences Research Council grant number GR/R43150, Formal Methods and Testing (FORTEST). We would like to thank the anonymous referees for their valuable comments; these significantly strengthened the paper.

References

[1] H. AboElFotoh, O. Abou-Rabia, and H. Ural. A test generation algorithm for protocols modeled as non–deterministic FSMs. *The Software Engineering Journal*, 8(4):184–188, 1993.

[2] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer–Verlag, 2001.

[3] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[4] P. Eades and X. Lin. A heuristic for the feedback arc set problem. *Australasian Journal of Combinatorics*, 12:15–25, 1995.

[5] G. Even, J. S. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multi–cuts in directed graphs. *Algorithmica*, 20:151–174, 1998.

[6] P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. Technical Report 99.2.2, AT&T Labs, 1999.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, New York, 1979.

[8] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3):270–285, 1993.

[9] R. M. Hierons. Adaptive testing of a deterministic implementation against a nondetermistic finite state machine. *The Computer Journal*, 41(5):349–355, 1998.

[10] R. M. Hierons. Testing from a non–deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

[11] R. M. Hierons and H. Ural. Concerning the ordering of adaptive test sequences. In *23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2003)*, volume 2767 of *Lecture Notes in Computer Science*, pages 289–302, Berlin, Germany, September 29 – October 2 2003. Springer–Verlag.

[12] Joint Technical Committee ISO/IEC JTC 1. *International Standard ISO/IEC 9646-1. Information Technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts.* ISO/IEC, 1994.

[13] ITU-T. *Recommendation Z.500 Framework on formal methods in conformance testing.* International Telecommunications Union, Geneva, Switzerland, 1997.

[14] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations.* Plenum Press, New York–London, 1972. 85–103.

[15] D. Lee and M. Yannakakis. Principles and methods of testing finite–state machines – a survey. *Proceedings of the IEEE*, 84(8):1089–1123, 1996.

[16] C. L. Lucchesi. *A minimax equality for directed graphs.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1976.

[17] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, 1996.

[18] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong. Empirical studies of test-suite reduction. *Journal of Software Testing, Verification and Reliability*, 12(4):219–249, 2002.

[19] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2), 1972.

[20] J. Tretmans. Conformance testing with labelled transitions systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29(1):49–79, 1996.

[21] P. Tripathy and K. Naik. Generation of adaptive test cases from non–deterministic finite state models. In *Proceedings of the 5th International Workshop on Protocol Test Systems*, pages 309–320, Montreal, September 1992.

[22] C. Willcock, T. Deiss, S. Tobies, S. Keil, F. Engler, and S. Schulz. *An Introduction to TTCN-3.* Wiley, 2005. New Jersey.

[23] S. Yoo, M. Kim, and D. Kang. An approach to dynamic protocol testing. In *IFIP TC6 10th International Workshop on Testing of Communicating Systems*, pages 183–199, Cheju Island, Korea, September 1997. Chapman and Hall, London.