Minimizing the cost of Fault Location when testing from a Finite State Machine

Robert M. Hierons
Department of Mathematical and Computing Sciences, Goldsmiths College, University of London
Email: r.hierons@gold.ac.uk

Abstract
If a test does not produce the expected output, the incorrect output may have been caused by an earlier state transfer failure. Ghedamsi and von Bochmann [1992] and Ghedamsi et al. [1993] generate a set of candidates and then produce further tests to locate the failures within this set. We consider the special case where there is a state identification process that is known to be correct. A number of preset and adaptive approaches to fault location are described and the problem of minimizing the cost is explored. Some of the approaches lead to NP-hard optimization problems for which possible heuristics are suggested.
Keywords: Finite state machine, fault location, adaptive testing, minimal length test.

1. Introduction

The use of *Finites State Machines* (FSM) to model systems has lead to much interest in deriving tests from them (e.g. Aho et al. [1988], Petrenko et al. [1994b], Hierons [1996], Ural [1997], and Hierons [1997]). Most work has concentrated on developing tests with certain properties and, possibly, finding the shortest such test. Section 2 will provide a brief introduction to testing from FSM.

If an incorrect output, called a *symptom*, is detected during testing it may have been caused by either an incorrect output (an *output fault*) or an earlier incorrect state transfer (a *state transfer fault*). It is therefore important to apply a strategy to locate the failure that occurred.

Ghedamsi and von Bochmann [1992] and Ghedamsi et al. [1993] generate a set of transitions whose failure could explain the behaviour exhibited (these transitions are called *candidates*). They then produce tests (called *distinguishing tests*) in order to find the faulty transitions within this set. The problem of generating candidates is briefly considered, in the situation in which there has been only one failure, in Section 3.

In this paper the special case, where there is a state verification process that is known to be correct, will be investigated. Algorithms, that generate distinguishing tests with low order polynomial size, will be given for both single and multiple faults.

Several algorithms for generating distinguishing tests, in the presence of one fault, will be considered in Section 4. In Section 5 the problem of minimizing the cost is discussed. Some of these optimization problems are NP-hard and p heuristics are discussed in Section 6.

While the algorithms given assume that the state verification process is correct, sometimes variants of them can be used when the state verification process contains candidates. This is examined in Section 7.

The problem of fault location in the presence of multiple faults is discussed in Sections 8 and 9. It transpires that many of the approaches discussed for single faults can be extended to this more general case. While Ghedamsi et al. [1993] provide a general solution to the problem of fault location, their algorithm suffers from combinatorial explosion and produces a distinguishing test whose size is exponential

(in the number of transfer faults and the number of test cases with symptoms). We show that this combinatorial explosion can be avoided if there is a state verification process that is known to be correct. They also assume that all faults have been reached: we introduce one approach that does not assume this. Finally, in Section 10, conclusions are drawn.

2. Preliminaries

2.1. Finite state machines

A *finite state machine* (also called a *Mealy Machine*) is defined by a tuple $(S,T,s_0, ,)$ where S is a finite set of states, T is a finite set of transitions between states, s_0 is the initial state, is the finite input alphabet, and is the finite output alphabet. Each transition is defined by a triple $(s_i,s_j,in/out)$ in which s_i is the initial state of the transition, s_j is the final state of the transition, and *in/out* is the input/output behaviour. A graphical representation of the FSM given in Aho et al. [1988] is shown in Figure 1. This FSM will be denoted G throughout this paper. A general FSM with n states will be denoted $M=(S,T,s_0, ,)$.

A number of systems can be modelled by FSM, an example being the control section of a communications protocol (see e.g. Huang and Hsu [1994]). It has been noted that many Process Algebra specifications can be converted into an FSM (Petrenko et al. [1994a]).

An FSM is said to be *deterministic* if there are no pairs of transitions that have the same initial state and input. The transitions of a deterministic FSM can be represented by a pair of functions and . The function gives the next state while gives the output: if the system is in state s_i and a transition with input *in* is executed the system moves to state (s_i,in) and produces output (s_i,in). These functions can be extended in a natural way by taking their closures, getting functions * (type S * S) and * (type S * *).

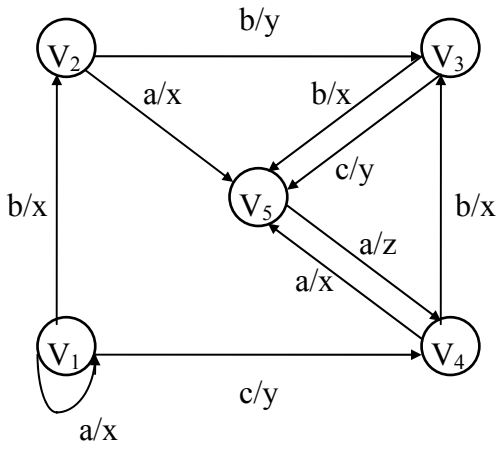

Figure 1

An FSM is said to be *completely specified* if for each input *in* and each state s_i there is a transition t_j from s_i that has input *in*. If an FSM F is not completely specified, a completely specified FSM F' can be produced from F by either adding an *error state* or by adding *null actions*. An example is the FSM G in Figure 1 which

can be converted to a completely specified FSM *G'* by adding the transitions (V_2,V_2,*c/null*), (V_3,V_3,*a/null*), (V_4,V_4,*c/null*), (V_5,V_5,*b/null*) and (V_5,V_5,*c/null*).

Two states of an FSM F are said to be *equivalent* if and only if they have the same set of valid input sequences and for each allowed input sequence they allow the same set of output sequences. If an FSM *F* has no pair of equivalent states it is said to be *minimal*.

A *directed graph (digraph)* is defined by a set of vertices and a set of directed edges between the vertices. Labels can be added to the edges. Thus, an FSM can be represented by a digraph in which the vertices correspond to states and the edges, labelled by the input/output, correspond to the transitions.

An FSM is *strongly connected* if, given any ordered pair of states (s_i,s_j), there is a sequence of transitions that will move the FSM from s_i to s_j. See e.g. Kohavi [1978] for more information on FSM. It will be assumed throughout this paper that any FSM considered is deterministic, minimal, completely specified, and strongly connected.

2.2. State verification

In order to check the final state of a transition it is necessary to further execute the system. There are three main strategies (see e.g. Sidhu and Leung [1988] or Cavalli et al. [1994]) which use:

1. Distinguishing sequence (DS)
2. Unique Input/Output (UIO)
3. Characterizing set

A *distinguishing sequence* is an input sequence that produces a different output for each state. It is easy to check that the FSM *G'* has a number of distinguishing sequences, including *ba*.

A *unique input/output sequence X* for a state s_i is an input sequence that has the property that, for all s_j s_i $^*(s_i,X)$ $^*(s_j,X)$. Although *X* can verify s_i it need not be able to verify any other state. Clearly a DS is a UIO for each state. It is noted in Kohavi and Kohavi [1968] that it is possible to use an adaptive DS if it is not necessary to produce a preset test sequence and, similarly, adaptive UIOs could be used.

Some FSM do not have either a DS or a UIO for each state. Every minimal FSM does, however, have a *characterizing set* (see e.g. Chow [1978]). A characterizing set *W* is a set of input sequences with the property that, if every sequence is executed from some state s_i, the set of output sequences verifies s_i. It is noted by Fujiwara et al. [1991] that for some states not every element of *W* is required: some subset can be used (the Wp method). This can reduce the effort involved in verifying a state.

In this paper it will be assumed that each state of any FSM used has a UIO that is known to be correct. This knowledge may have been derived from testing. The approaches outlined can, however, be extended to the case where a characterizing set is used.

2.3. Conformance testing

When testing from an FSM model it is assumed that the *implementation under test* (*IUT*) can be modelled by an FSM and thus that testing involves comparing the

behaviour of two FSM. Often a failure can be categorized as either an output failure or a state transfer failure. Naturally, this categorization requires a close correspondence between the states of the two FSM (the existence of a valid state identification process makes this categorization clear).

It is possible to test a transition by moving to the initial state of the transition, applying the transition, and then checking the final state (e.g. Chow [1978]). Aho et al. [1988] instead looked at the question of how *test subsequences*, of the form of a transition followed by a UIO, can be combined in a manner that minimizes the length of the total test sequence. They describe the problem as an instance of the *Rural Chinese Postman Problem (RCPP)*. The RCPP is the problem of producing a minimal tour that passes through some given set of edges, in this case edges that represent test subsequences.

While the RCPP is plete (see Lenstra and Kan [1976]), Aho et al. [1988] apply a polynomial algorithm that produces an optimal solution under certain conditions. Two sufficient conditions for this algorithm to work are that either the FSM has a *reset operation* (some input that takes every state to the initial state) or that the FSM has a *loop* (some transition with the same initial and final state) for each state. While these conditions are sufficient they are not necessary.

Yang and Ural [1990] note that the test subsequences may overlap and utilize this to reduce the test sequence length. Hierons [1996] represent this overlap using invertible transitions, where a transition $(s_i,s_j,x/y)$ is *invertible* if it is the only transition entering state s_j with input x and output y. Invertibility is extended to sequences by Hierons [1997].

3. Single Faults: Generating Candidates

3.1. Introduction

In this section it will be assumed that the IUT has only one fault. The results will be generalized to multiple faults in Sections 8 and 9. For the rest of this paper it will be assumed that some test set $TS=\{tc_1,...,tc_p\}$, where the tc_i are test cases, has been executed. A test case tc_i consists of a sequence of expected transitions $t_{i,1},...,t_{i,m(i)}$, starting at s_0, with input values $x_{i,1},x_{i,2},...,x_{i,m(i)}$ and expected output $y_{i,1},y_{i,2},....,y_{i,m(i)}$. When executed, tc_i produces actual output $z_{i,1},z_{i,2},...,z_{i,m(i)}$. A symptom is any i,j such that $y_{i,j}$ $z_{i,j}$. As tests after a symptom will be ignored, without loss of generality it will be assumed that symptoms occur in the final transitions of test cases only. The approach outlined in this section is similar to that given in Ghedamsi and von Bochmann [1992].

If the input/output of a symptom is not allowed by any state, there must have been a failure in this execution, but otherwise an earlier state transfer failure may have lead to an incorrect state. It is thus necessary to develop the set of transitions whose failure could explain the behaviour exhibited.

If a system that was intended to implement the FSM *G'*, defined earlier, was tested with the input sequence *abaab* and produced the output sequence *xxxzy*, rather than the expected output sequence *xxxzx*, this situation would exist. The transition *b/y* is valid from some states, though not the expected state. There are therefore two possibilities: either the last execution was incorrect or some previous transition lead to an incorrect state.

3.2. Generating Conflict Sets

For a symptom, at input $x_{i,m(i)}$ in test case tc_i, there is an initial candidate list consisting of $\{x_{i,j} | 1 \ j \ m(i)\}$. Strategies that can reduce this set without further testing will now be considered.

If some subsequence of the observed behaviour is not allowed from any state of the model there must have been at least one failure in this subsequence. The smallest such subsequence is found. This is defined by the largest m giving input/output behaviour $(x_{i,m}/z_{i,m},...,x_{i,m(i)}/z_{i,m(i)})$, and corresponding set of transitions $\{t_{i,m},....,t_{i,m(i)}\}$, that is not allowed from any state. As there is only one fault, one of these transitions was erroneous. This set is called a *conflict set*.

In the earlier example, where output *xxxzy* was produced from input *abaab*, it is possible to apply this approach. In this case there is no state in the model from which the sequence *a/z,b/y* is valid and therefore there must be a failure in at least one of these transitions. This reduces the search space for the failure to 2 of the 5 transitions.

3.3. Generating Diagnostic Candidates

The conflict sets can now be considered as a group. An initial tentative candidate set is given by the intersection of the conflict sets.

If only one transition, t_o, provides symptoms then this is a possible candidate. The tests are checked to see whether they are consistent with t_o being the fault: if all occurrences of t_o produce the same output then t_o is included in the set of candidates.

If there is more than one transition that produces a symptom then, as there is only one fault, the fault must be a state transfer fault. For every transition t_i that may have a transfer fault the set, $EndStates_i$, of possible alternative final states for t_i is found. The final set of candidates, T_E, is the set of the t_i for which $EndStates_i$ $\{\}$ and possibly a transition that may have an output fault. Algorithms for finding these sets are to be found in Ghedamsi and von Bochmann [1992].

4. Locating Single Faults

4.1. Overview

Some set, T_E, of candidates $t_1,...,t_q$ has been found. In order to locate the fault it is necessary to further execute the system. Two main strategies that will be considered:

1. Executing subsequences $t_{i,1},...,t_{i,k}$ in each case verifying the final state.
2. Individually testing the transitions from T_E.

These approaches will now be discussed and minimizing the cost, when using the second approach, will be discussed in more detail in Section 5. It should be noted that Ghedamsi and von Bochmann [1992] and Ghedamsi et al. [1993] do not consider minimizing the cost of fault location.

4.2. Executing subsequences

Given a test case tc_i that contains a symptom at transition $t_{i,m(i)}$ and a number of possible candidates $t_{i,m},...,t_{i,m(i)}$ in T_E it is possible to develop tests by checking the final state of subsequences of $t_{i,m},...,t_{i,m(i)}$. Suppose s is the initial state of $t_{i,m}$. One approach is to initially execute $(x_{i,m},...,x_{i,m(i)-1})$ from s and check the final state, if it is possible for there to have been an output fault. If this is not the expected state there

has been a failure in this subsequence and so $(x_{i,m},...,x_{i,m(i)-2})$ is executed from s and its final state checked. This can be repeated until a failure has been isolated. As there is only one fault, it is sufficient to apply this to any test case that contains a symptom.

This process is like searching an ordered list of length $O(m(i)-m)$. Suppose T is an upper bound on the test sequence length. If UIOs are used and the length of these is bounded above by u then the test length is of $O((T+u)(m(i)-m))$. If a characterizing set is used, this has $O(n)$ elements each of $O(n)$ length and thus the test length is of $O(n(m(i)-m)(n+T))$.

Searching an ordered list in a linear manner is relatively inefficient. This suggests the use of an *adaptive* process in which a binary search is used. The first step is to check the final state produced by executing $(x_{i,m},...,x_{i,m(i)-1})$ from s, if there is a possible output fault. At each subsequent stage the state of the midpoint of the subsequence being considered is checked. If this is correct, the second half of the subsequence is now considered and otherwise the first half is used. This proceeds until the subsequence being considered has length one: the fault has been located.

If UIOs are being used the size of the test set is of order $O((u+T)\log_2(m(i)-m))$ as there are $O(\log_2(m(i)-m))$ steps. If a characterizing set is being used the test size is of $O(n(n+T)\log_2(m(i)-m))$.

If there is a state identification process that is known to be correct and a reliable reset, this approach to locating the failure can *always* be applied. This approach may therefore be particularly useful in cases where it is difficult to test the individual transitions without using transitions from T_E.

4.3. Individually testing the elements of T_E

If T is large relative to n, testing subsequence may be inefficient as it is necessary to execute a subsequence with length $O(T)$ in order to reach a transition to be tested. Instead, to test a transition t_i T_E it is possible to use a sequence that takes the system to $head(t_i)$, executes x, and then executes a UIO for $tail(t_i)$. This process must avoid other (untested) candidates. If there is a possible output fault it is natural to initially test this, but if this is not erroneous some approach to testing the other transitions is required. The following strategies will be considered:

1. testing the transitions separately, each time resetting after the test (*testing with reset*).
2. producing one sequence containing tests for every candidate.

The first approach is similar to that given in Ghedamsi and von Bochmann [1992], though they do not consider the problem of minimizing the cost. The optimization problem is simpler for the former, and will be discussed in Section 5. The optimization problem for the latter will be discussed in Section 5 and further developed in Section 6.

5. The Cost of Locating Single Failures

5.1. Introduction

It will be assumed throughout the rest of Sections 5 and 6 that there is a state transfer fault (if there is a possible output fault it has been checked). The cost of testing candidates separately will be investigated.

There are two possible costs to consider: the worst case and the expected case. Both will be investigated but, as the expected case requires the existence of a probability distribution, the generation of such a distribution will first be discussed.

5.2. Assigning a probability

For the rest of this section it will be assumed that the only knowledge present, that can be used to derive a probability, is the behaviour exhibited. If there is other information, such as it being known that certain parts of the system are more likely to contain faults than others, this can be used: the rest of the paper only assumes that some probability of failure has been assigned to each t_i T_E.

The mutant FSMs, that are consistent with the behaviour exhibited and differ only in one state transfer, can be found. Unless there is extra information it will be assumed that these mutants are equiprobable. Thus, for a candidate t_i, the proportion of mutants that have t_i as a failure gives the probability that t_i is erroneous.

The number of mutants is proportional to the number of *alternative final states* for t_i that allow the observed behaviour. The probabilities, p_i, can be defined by:

$$p_i = \frac{|EndStates_i|}{\sum_{t_i \ T_E} |EndStates_i|}$$

If there is additional information, it is possible to adjust these values.

5.3. Ordering testing with reset

5.3.1. Determining the optimal ordering

When testing with reset the worst case cost varies very little with the order of testing as it is effectively the cost of testing every transition. The expected effort will thus be considered.

For the purposes of optimization, the cost of testing a transition can be considered to be zero if the transition is erroneous, as this transition must be tested in order to find the failure. The cost is t (probability $(1-p_i)$), in which case the cost is given by the length of the sequence required to test t_i, which will be denote $c(t_i)$. If the shortest path, avoiding other candidates, to the initial state of t_i is denoted $path(t_i)$ and the UIO used for the final state of t_i is u_i then:

$$c(t_i) = |path(t_i)|+|u_i|+2$$

If $head(t_i)$ cannot be reached while avoiding other candidates, $c(t_i)$ is infinite. The expected cost of testing t_i is given by:

$$C(t_i)=(1-p_i)c(t_i)$$

The transitions are tested in order of *increasing* $C(t_i)$. If it is only necessary to avoid untested candidates, these values can be updated after each test, as the $|path(t_i)|$ can change. Clearly there will always be an order that allows untested transitions to be avoided, as the UIOs are known to be correct.

5.3.2. Determining the expected cost

Sometimes it is useful to have an estimate of the cost of fault location. In order to produce an *expected cost of testing*, the expected cost of testing an erroneous

transition is required. Given some t_i T_E an estimate of the cost of testing t_i is produced by, for each x $EndStates_i$, determining the cost of testing t_i if the final (erroneous) state actually is x. Given x $EndStates_i$ and UIO $u_k=a_1,...,a_r$ for $tail(t_i)$, the expected number of transitions required to distinguish x from $head(t_i)$ is given by:

$ident(tail(t_i),x,u_i) = min\{b|\ ^*(x,a_1,...,a_b)\ \ ^*(tail(t_i),a_1,...,a_b)\}$

Thus the expected number of transitions required to determine that the final state of t_i is erroneous is:

$$Fail(t_i,u_i)\quad \frac{\sum_{x\ EndStates_i} ident(tail(t_i),x,u_i)}{|EndStates_i|}$$

Let the set U give a UIO for each state and contain UIO u_i for $tail(t_i)$. Then the expected cost of testing t_i, if t_i is erroneous, is given by:

$D(t_i,U) = |path(t_i)|+1+ Fail(t_i,u_i)$

If the transitions are tested in order , which is a permutation of $(1,...,q)$, and the incorrect transition is $t_{(i)}$, the following gives the expected cost of testing:

$$E(,i,U) = D(t_{(i)},U) + \sum_{j\ 1}^{i\ 1} c(t_{(j)})$$

Thus, given permutation and UIO set U, the expected cost of testing in the order is:

$$EC(,U) = \sum_{i\ 1}^{q} p_{(i)}E(,i,U)$$

5.4. Testing without reset

5.4.1. Overview

When testing without reset the test is made up of test subsequences, each corresponding to some t_i T_E followed by a UIO, connected to form a single test sequence. The worst case and expected case costs will be investigated.

5.4.2. Worst and expected case analysis

It will be assumed that if one of the candidates is an output fault then this has been tested and removed from T_E and that by removing the candidates from M we do not disconnect M.

Given a valid permutation the problem of finding a valid test sequence can be broken down into a number of sub-problems, one for each $t_{(i)}$. These sub-problems are of the form: test the transition $t_{(i)}$ while avoiding (untested) candidates. The test sequence required to test the transition $t_{(i)}$ with UIO set U, and in order shall be denoted $test(,t_{(i)},U)$. This represents the shortest route from the final state of the UIO used for $t_{(i-1)}$ to the initial state of t (i) followed by $t_{(i)}$ followed by some UIO $u_{(i)}$. All this must avoid the set of elements from T_E that are untested.

For each UIO, $test(,t_i,U)$ can be produced in $O(n^2)$. The cost of testing the first i transitions, if they are correct, in the order given by is:

$Cost(,i)= _{1 j i}|test(,t_{(j)},U)|$

The worst case analysis is given by this with $i=q$, adjusted to account for the fact that the final transition is erroneous. Thus the worst case is given by:

$WC(,U) = Cost(,q)+Fail(t_{(q)},u_{(q)})-|u_{(q)}|$

This can be calculated in $O(n^2|T_E|)$. The problem of finding the optimal order is, however, potentially much more difficult. This problem is simplified if *all* candidates are avoided throughout testing: the problem is then an instance of the RCPP. Possible heuristics for the general case are discussed in Section 6. When the problem is an instance of the RCPP it may be possible to use any overlap that exists between the test subsequences, possibly using approaches similar to those described in Yang and Ural [1990], Hierons [1996], and Hierons [1997].

The expected cost of testing may also be considered. The expected cost is given by:

$$ECost(,U) = \sum_{i=1}^{q} p_{(i)}(Cost(,i)+Fail(t_{(i)},u_{(i)})-|u_{(i)}|)$$

This can be calculated in $O(n^2|T_E|)$. The problem of producing the order that minimizes the expected cost is therefore: find a permutation of $(1,...,q)$ that minimizes $ECost(,I)$. Again, possible heuristics are discussed in Section 6.

5.5. Test Length

When testing with one test sequence or testing using reset, each subsequence length is of $O(n+u)$ if UIOs are used. If a characterizing set is used, the subsequences are all of length $O(n)$ and thus the subsequences for a single candidate have length of $O(n^2)$. When using UIOs the overall test length is of $O(|T_E|(n+u))$ and when using a characterizing set the overall test length is of $O(|T_E|n^2)$.

It should be noted that these orders apply even if there is no attempt to find the optimal permutation. Given any permutation, , a test of this order can be found in $O(n^2|T_E|)$ time.

6. Heuristics to minimize the expected cost

Given a permutation the problem of calculating the expected and worst case costs are of $O(|T_E|n^2)$. Clearly, it is not usually feasible to calculate this for every permutation.

Two cases will be considered: either avoiding T_E or avoiding untested elements of T_E in testing. The latter may produce shorter test sequences but is more complex as the allowed connecting paths between subsequences depends upon the position within the test.

A number of heuristics, such as *Tabu search* and *Hill Climbing*, are based upon making an initial guess and then making a number of small changes. The initial guess would be some permutation and the small changes would be in the form of

swapping adjacent elements. For each swap, the change in expected cost is calculated. There are two possible approaches to generating the change in expected cost:

1.Generating these separately.

2.Initially generating a matrix of distances in $(S,T\backslash T_E,s_0, ,)$) and calculating the costs from this.

The second approach can only be applied when avoiding the use of all candidates in testing and may thus be suboptimal.

The first approach gives complexity $O(n^2)$ for each swap. The second approach gives an initial effort of $O(n^3)$ (see e.g. Gibbons [1985]) but then each swap has complexity $O(1)$. If, as is usual, many swaps are required, the second approach may be preferable.

7. State Identification Sequences containing elements of T_E

Some states may not have UIOs that are known to be correct. This adds a dependency between the transitions: it may be necessary to check some set of transitions before testing a transition t_i, as these are used in the UIO for t_i. The following algorithm determines whether there is some order that allows this:

Algorithm

Let $T_E=T\backslash T_E$
While T_E {}
Find some t_i in T_E that can be tested using transitions from T_E'. If there is not such t_i then Fail.
Remove t_i from T_E and add it to T_E'
End {while}
Succeed

A permutation of $1,...,q$ will be said to be *valid* if, for every i,j such that $1 \quad (i) < (j) \quad q$, the UIO for $t_{(i)}$ does not contain $t_{(j)}$. If heuristics, that involve swapping adjacent elements in the permutation, are used it is important that the allowed permutations are connected by valid swaps. The following result shows that this is the case when $(S,T\backslash T_E,s_0, ,)$ is strongly connected:

Lemma

Suppose $M'=(S,T\backslash T_E,s_0, ,)$ is strongly connected. Then given valid permutations $_1$ and $_2$ on $(1,...,q)$ it is possible to move from $_1$ to $_2$ via a sequence of valid permutations $_1= _0', _1',..., _r'= _2$ such that for any j, $_j'$ and $_{j+1}'$ differ only by two adjacent elements being swapped.

Proof

Proof will be by induction on q. The result is trivially true for the base case of $q=1$. The inductive hypothesis is: the result holds for all cases in which $q=k$. It is sufficient to prove the result holds for any valid pair of permutations $(_1, _2)$ of $(1,...,q)$ such that $q=k+1$.

If $_1(1)= _2(1)$ the result follows from the inductive hypothesis.

If $_1(1)$ $_2(1)$ it is sufficient to prove that $_1$ can be transformed into some valid permutation $_1'$, via a sequence of valid permutations that differ by swaps of adjacent elements, such that $_1'(1) = _2(1)$.

In order to do this the element $_2(1)$ can be located in $_1$ and, as this element appears at the beginning of a valid permutation $_2$, it has a state identification sequence that avoids all elements of T_E. It can therefore be swapped with the preceding element of $_1$ to produce a valid permutation, as it is not necessary to test this previous element in order to test $_2(1)$ and it is always possible to connect states without using transitions from T_E.

Thus $_2(1)$ can be repeatedly moved forward via individual swaps in this manner, each time producing a valid permutation, until there is some permutation $_1'$ such that $_1'(1) = _2(1)$. The inductive hypothesis can now be applied to $(_1', _2)$ and the result follows.

□

It should be noted that *M'* being strongly connected is a sufficient, but not necessary, condition.

These results suggest that heuristics such as Hill Climbing and Tabu Search are appropriate where there is some feasible solution.

8. Multiple Faults

8.1. Overview

Ghedamsi et al. [1993] consider the problem of finding a set of candidates, and then distinguishing tests, in the case where there may be more than one fault. The process of finding a set of candidates is similar to that used with single failures. Ghedamsi et al. [1993] also assume that every fault is reached by a test: each one is contained in a test sequence in which it is not preceded by a fault and in which it leads to a symptom. Some of the approaches we discuss require this condition.

8.2. Generating Conflict sets

These can be developed in very much the same way as when there is a single fault. The main difference is that there may be more than one fault met by a test case and thus, while a minimal subset whose behaviour is not allowed can be found, the initial state of this may not be that expected. In order to check for this it is possible to include an initial test to check the final state of the preceding sequence.

Suppose for some tc_i the value m gives the shortest subsequence (finishing at the symptom $t_{i,m(i)}$) that is not allowed from any state. If the final state of $t_{i,1},...,t_{i,m-1}$ is as expected, one of the transitions from $t_{i,m},...,t_{i,m(i)}$ is erroneous. If the final state of $t_{i,1}...t_{i,m-1}$ is not that expected, there is some corresponding set of transitions (starting at the actual final state of $t_{i,1}...t_{i,m-1}$) that contains at least one failure. This suggests executing a further test to determine the final state of $t_{i,1},...,t_{i,m-1}$.

From now on it will be assumed that the final state of $t_{i,1},...,t_{i,m-1}$ is correct and thus that it is know that there is a fault in $\{t_{i,m},...,t_{i,m(i)}\}$: if this is not the case $t_{i,m},...,t_{i,m(i)}$ is simply replaced by the transitions expected from the actual final state of $x_{i,1}...x_{i,m-1}$ and the candidate set $\{t_{i,1},...,t_{i,m-1}\}$.

If some test case tc_i does not contain a failure in its first output, there is no output fault for the corresponding transition. All such output faults can be removed from the set of candidates.

8.3. Generating Diagnostic Candidates

Ghedamsi et al. [1993] take the set of tentative candidates and consider their consistency. They generate the mutant machines (with the same number of states as M) that are consistent with the behaviour observed. Unfortunately a combinatorial explosion can occur at this step.

We propose an alternative (adaptive) approach: for each test case tc_i that contains a symptom, test the candidates from this test case while avoiding all other candidates. Once this has been done for a test case, update the candidates within the other test cases and repeat the process. This continues until all failures are explained.

9. Minimal Effort with Multiple Faults

9.1. Ordering

Approaches equivalent to those outlined in Sections 5 and 6 can be used to diagnose the fault in a test case that contains a symptom. This process can be repeated for all test cases containing symptoms. It is certainly the case that, when there is a correct state verification process, testing using subsequences and testing with reset can *always* be applied to the individual test cases that contain symptoms.

Testing with subsequences has the additional advantage that, unlike the algorithm given in Ghedamsi et al. [1993], it does not rely on the assumption that all faults have been reached. It can thus be applied in some situations in which the approach given in Ghedamsi et al. [1993] is not appropriate.

The order in which test cases are tackled can affect the cost, as the diagnosis for one test case can reduce the effort required for another. Thus, for example, if a transition is found to be correct it can be removed from the candidate sets for other test cases.

An order that optimizes the expected saving, derived from removing candidates from other test cases, is required. Let tcc_i denote the set of candidates in tc_i. Without loss of generality, the set of test cases containing candidates is $tc_1,...,tc_k$. Let J denote the index set $1..k$ and $TC_i = \bigcup_{j \in J\setminus\{i\}} tcc_j$. Then $w_i = |TC_i - tcc_i|$ denotes the number of candidates from tc_i that are contained in other candidate sets.

A natural heuristic is to diagnosed in order of *decreasing* w_i. The value of each w_i must, however, be updated after every test and thus the test order is adaptive. For example, one diagnosis for a test case tc_i may simply remove one candidate from tc_j but a different diagnosis for tc_i may fully diagnose tc_j.

9.2. Test Length

Suppose there is an upper bound T on the length of the test cases, m_t state transfer faults, and m_o output faults. Let $E=m_t+m_o$.

Initially we will consider testing using subsequences. Then using a linear search the test length is of $O((T+u)(m_tT+m_o))$ when using UIOs and using a characterizing set the test length is of $O((T+n)n(m_tT+m_o)$. Alternatively, when using a binary search the test length is of $O((T+u)(m_t\log_2(T)+m_o))$ if UIOs are used and $O((T+n)n(m_t\log_2(T)+m_o)$ if a characterizing set is used.

Let us now consider testing candidates separately. Each test case that contains a transfer fault requires $O(T(n+u))$ transitions if UIOs are being used and $O(Tn^2)$ if a characterizing set is being used. Thus the overall diagnostic test length is

$O((m_o+m_tT)(n+u))$ when UIOs are being used and $O((m_o+m_tT)n^2)$ when a characterizing set is being used.

10. Conclusions

If an incorrect output has occurred it may have been caused by a previous state transfer failure. If this is the case, further tests must be performed in order to determine which transition failed. There are then two cases to consider: it is known that there is only one fault or there may be more than one fault.

Ghedamsi et al. [1993] produce a general solution to the problem of fault location with multiple faults, but the length of the test generated is exponential in the number of transfer faults and the number of test cases with symptoms. Instead we consider the situation in which there is a correct UIO for each state. Algorithms have been introduced that generate tests with (low order) polynomial lengths. These can easily be generalized to the use of a characterizing set.

It is desirable to minimize the effort involved in fault location and the problems of minimizing the expected case and worst case have been discussed. While some of these problems are NP-hard, they have proper
of heuristics such as Tabu Search and Hill Climbing.

Two basic approaches are described: testing subsequences and testing individual transitions. The latter approach should produce shorter tests but relies on the assumption that all faults have been met (Ghedamsi et al. [1993] also makes this assumption). The decision as to which approach should be used may thus depend upon the technique used to generate the initial test.

The expected case analysis requires the existence of a probability distribution. The tester may have knowledge of the system that allows the generation of such a distribution. Alternatively a method based on determining the number of possible next states, for each possible state transfer fault, can be used. If a uniform distribution is used, the expected case and worst case problems are equivalent.

If some UIOs contain candidates there is a dependency between the candidates and the problem becomes more complex. An algorithm is given, for the case where there is one fault, that determines whether there is a test order that is consistent with these dependencies. If such an order exists heuristics such as Hill Climbing and Tabu Search may be appropriate.

11. References

1. Aho A.V., Dahbura A.T., Lee D., and Uyar M.U. 1988. An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours, Proceedings of *Protocol Model, Testing, and Verification VIII*, IFIP 1988, pp75-86
2. Cavalli A.R., Favreau J-P, al Methods in Philippo Conformance Testing: Results and Perspectives, Proceedings of *Protocol Test Systems VI*, IFIP, pp3-17
3. Chow T.S. 1978. Testing Software Design Modelled by Finite State Machines, *IEEE Transactions on Software Engineering*, **4** 3 March 1978 pp178-18
4. Fujiwara S., Bochmann G., Khendek F., Amalou M., and Ghedamsi A. 1991. Test Selection Based on Finite State Models, *IEEE Transactions on Software Engineering*, **17** 6 June 1991, pp591-603
5. Gibbons A. 1985. *Algorithmic Graph Theory*, Cambridge University Press.

6. Ghedamsi A. and Von Bochmann G. 1992. Test Result Analysis and Diagnosis for Finite State Machines, Proceedings of the *12th International Workshop on Protocol Test Systems*, June 9-11, Yokohama, pp 244-251
7. Ghedamsi A., Von Bochmann G., and Dssouli R. 1993. Multiple Fault Diagnosis for Finite State Machines, Proceedings of *IEEE INFOCOM'93*, pp 782-791.
8. Hierons R.M. 1996. Extending Test Sequence Overlap by Invertibility, *The Computer Journal*, **39** 4, pp 325-330
9. Hierons R.M. 1997. Testing From A Finite State Machine: Extending Invertibility to Sequences, *The Computer Journal*, **40** 4
10. Huang C.-M. and Lin M.-S. 1994. An Incremental Protocol Verification Method, *The Computer Journal*, **37** 8, pp 698-710.
11. Kohavi Z. 1978. *Switching and Finite Automata Theory*, New York: McGraw-
12. Lenstra J.K. and Rinnooy Kan A.H.G. 1976. On General Routing Problems, *Networks*, **6** pp 273-280.
13. Petrenko A., von Bochmann G., and Dssouli R. 1994a. Conformance Relations and Test Derivation, Proceedings of *Protocol Test Systems*, VI **C-19**, pp 157-178
14. Petrenko A., Yevtushenko N., Lebedev A., and Das A. 1994b. Nondeterministic State Machines in Protocol Conformance Testing, Proceedings of *Protocol Test Systems VI*, **C-19**, pp 363-378
15. Sidhu D. and Leung T. 1988. Test Generation for Real Experie
Protocols, ACM SIGCOMM 88, pp 257-261
16. Yang B. and Ural H. 1990. Protocol Conformance Test Generation Using Multiple UIO Sequences with Overlap, in *SIGCOMM 90, Communications, Architectures and Protocols*, Sept 24-27 1990, pp 118-125