Automatic Generation of Test Sequences form EFSM Models Using Evolutionary Algorithms

AbdulSalam Kalaji.
Robert M. Hierons.
Stephen Swift.
School of Information Systems, Computing and Mathematics
Brunel University
Uxbridge
UB8 3PH

Abstract

Automated test data generation through evolutionary testing (ET) is a topic of interest to the software engineering community. While there are many ET-based techniques for automatically generating test data from code, the problem of generating test data from an extended finite state machine (EFSMs) is more complex and has received little attention. In this paper, we introduce a novel approach that addresses the problem of generating input test sequences that trigger given feasible paths in an EFSM model by employing an ET-based technique. The proposed approach expresses the problem as a search for input parameters to be applied to a set of functions to be called sequentially. In order to apply ET-based technique, a new fitness function is introduced to cope with the case when a test target involves calls to a set of transitions sequentially. We evaluate our approach empirically using five sets of randomly generated paths through two EFSM case studies: for INRES and class 2 transport protocols. In the experiments, we apply random and the proposed ET-based which utilizes our new fitness function. Experimental results show that the proposed approach produces input test sequences that trigger all the feasible paths used with a success rate of 100%, however, the random technique failed in most cases with a success rate of 20.8%.

***Keywords*: Conformance testing, evolutionary testing (ET), EFSM, FSM, test derivation, test data generation.**

1. Introduction

Errors in software can cause undesired consequences and testing is therefore an important stage of the software development process. However, manual testing is an expensive and time consuming process as well as error-prone hence automation is desirable. Automated test data generation has been the subject of interest to many researchers in the last decade in order to develop efficient methods which can replace conventional manual methods [1-4].

When a system is implemented, there is a need to test whether the implementation agrees with the system specification. This is usually performed by conducting conformance testing which tries to find any differences between the behavior of an implementation under test (IUT) and its specification. Conformance testing treats the IUT as a black-box, where a tester has no information about the internal system structure and only I/O behavior is available.

In order to derive a test sequence from a system specification, a model that represents the specification is required. Finite state machine (FSMs) and extended finite state machine (EFSMs) are commonly used for the purpose of test sequence derivation [5]. An FSM can only

model the control part of a system; an extension is needed in order to model a system which has control and data parts, e.g., communication protocols. Such systems are usually represented using an EFSM model.

The FSM model has been widely studied and many methods are available which employ different techniques for the purpose of test data generation [6-9]. Nevertheless, automated test data generation from EFSM model is complicated by the presence of infeasible paths and is an open research problem [10, 11].

In an EFSM model, a given path can be classified as either infeasible or feasible. The existence of some infeasible paths is due to the variable interdependencies among the actions and conditions. If a path is infeasible, there is no input test data that can cause this path to be traversed. Thus, if such a path is chosen in order to exercise certain transitions, these transitions are not exercised even if they can be exercised through other feasible paths. While the feasibility of paths is undecidable, there are several techniques that handle them in certain special cases [10-13]. An analysis of these techniques is beyond the scope of this paper.

When testing from an EFSM model, there are several test strategies including: *state coverage, transition coverage, path coverage and constrained path coverage* which can be employed in order to generate a test suite [14]. Generally, a test suite can consist of either one or many test sequences. In the first case, a single test sequence is applied to the initial state of an IUT. Since there is only one test sequence, there is no need to bring the IUT back to its initial state. However, in the second case, a test suite requires the availability of a reset method which brings the IUT back to its initial state every time a test sequence is applied. Due to the nature of EFSM model, a test suite which consists of a single sequence is unlikely to provide a sufficient coverage [15].

Many techniques for generating test sequences from an EFSM produce a set of paths through the EFSM [5], [9-13] and [16-21]. This leaves us with the problem of finding test data for each feasible path. Thus an approach which automatically generates test data to trigger a feasible path can potentially enhance existing EFSM testing techniques.

A path test data is the set of input values to be applied to the interaction parameter fields of the transitions included in that path in order for it to be taken. According to [22], the process of finding such test data is complicated and requires human involvement because of two main constraints. First, the domain of the available test data for interactions parameter fields is relatively large; however, the suitable test data for a given path is just a selection of only a small subset of this domain. Second, this subset is further refined when a path's transitions have guards where the test data should be selected in order to satisfy these guards. Furthermore, when different test data are needed to exercise the same path for the purpose of thorough testing, the selection of input test data becomes even harder. To this end, the approach presented in this paper aims to address the problem described as:

Given: a feasible path in an EFSM model

Problem: find a set of input test data that can cause this path to be traversed.

The contributions of this paper are the following:

1- It proposes an approach that applies ET-based technique to EFSMs
2- It introduces a new fitness function that enables ET-based technique to be applied to functional testing.
3- The proposed fitness calculation method is evaluated by comparing it to the existing ET-

based fitness calculation.

4- The paper also empirically validates the efficiency of the proposed approach by applying the approach to the INRES initiator and class 2 transport protocols EFSMs.

The technique presented in this paper can be potentially incorporated with other available testing techniques that provide a set of paths through an EFSM model to satisfy a particular test criterion. We applied our approach to generate transition coverage test suites for two protocol case studies namely the INRES initiator [23] and the simplified version of Class 2 transport protocol [16, 24].

The rest of the paper is organized as follows: Section 2 provides background information, including an overview of related test generation methods and a description of evolutionary algorithm (EA) and evolutionary testing (ET). A new fitness function for EFSM path test data generation is given in Section 3. Approach validation and experiments are discussed in Section 4. Concluding remarks and our future work are in Section 5.

2. Preliminaries

2.1. The model

A finite state machine (FSM) is a *Mealy* machine, which has a finite set of states, inputs, and outputs. An output is produced upon state transition and this occurs when applying an input to the machine. An FSM model can successfully represent the control part of a system e.g., a telephone device, however, an extension is needed in order to model a system with control and data parts e.g., communication protocols. When extending a *Mealy* machine with internal variables, predicates, and operations we get an extended finite state machine (EFSM). The EFSM model is a 6-tuple [16] (S, s_0, V, I, O, T) where:

- S is the finite set of logical states
- $s_0 \in S$ is the initial state
- V is the finite set of internal variables
- I is the set of input declarations
- O is the set of output declarations
- T is the finite set of transitions

The transition $t \in T$ is represented by the 5-tuple (s_s, i, g, op, s_e) in which:

- s_s is the start state of t
- i is the input where $i \in I \cup \{\text{Nil}\}$
- g is the guard and is either Nil or is represented as a set of logical expressions given in terms of variables in V' where $\phi \neq V' \subseteq V$
- op is the sequential operation which consists of simple statements such as output statements and assignment statements
- s_e is the end state of t.

In an EFSM model, there is a set of variables. One variable in particular is used to represent the machine state and is called *state*, also referred to by *major state* in order to differentiate it from the other variables called *context variables*. The *state* variable is used to represent the state of a finite state machine e.g., idle, wait for connection, connection opened and so on, whereas other machine data such as port number, sequencing numbers, data to transfer, etc. are usually stored in context variables. A state transition occurs when one of the machine's transitions is

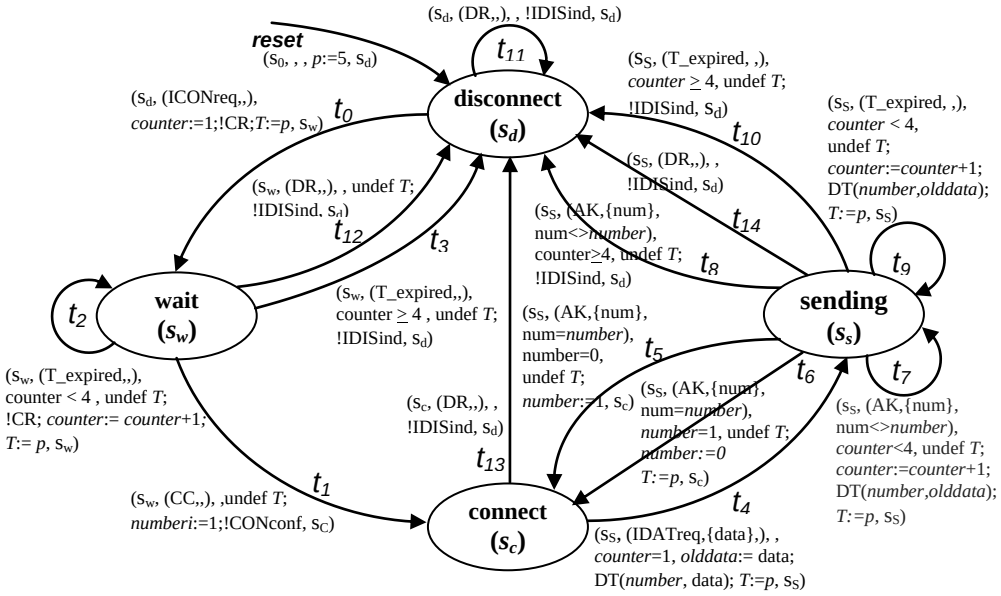

Fig.1 INRES protocol initiator as an EFSM

taken. Each transition has two major states: start state (S_s) and end state (S_e). Also, each transition may have one or more associated atomic operations to be executed upon a successful transition occurrence. These operations may update the value of some context variables or produce some output signals [20].

In order for a transition to be taken, there may be some required input values and associated conditions on context variables to be satisfied. According to this, EFSM transitions can be classified into two types: spontaneous and non-spontaneous transitions. Spontaneous transitions do not require input values in order to be taken; however, non-spontaneous transitions depend on some input value(s) in order to be initiated. Both spontaneous and non-spontaneous transitions can be partitioned to conditional and unconditional transitions depending on whether or not there exists one or more associated conditions, called guards, to be satisfied before a successful state transition can occur.

An EFSM is *deterministic* if for any group of transitions leaving the same state, it is not possible to satisfy the guards of more than one transition in this group at the same time and otherwise is *non-deterministic* [25].

The approach presented in this paper can be applied to an EFSM model with the following properties:

1- The specification represents a deterministic machine.
2- The specification does not include spontaneous transitions.

2.2. Examples

In this paper we present two EFSM case studies that we use in the experiments. For the purpose of avoiding repetition, we report only the EFSM definition of the second case study.

The first case study is the EFSM model of the INRES protocol initiator [23]. The INRES protocol is a connection- oriented and comprises the initiator, which establishes a connection and sends data, and the responder which receives data and terminates connections. INRES protocol has been designed to be similar to real protocols and yet small enough to allow conducting

Table 1. The core transitions in a class 2 transport protocol

t $s_s \rightarrow s_e$	Input declarations	Guards	Transition atomic operations
t_0 $s_1 \rightarrow s_2$	U?TCONreq(dst_add, prop_opt)	Nil	opt := prop_opt; R_credit :=0; N!TrCR
t_1 $s_1 \rightarrow s_3$	N?TrCR(peer_add, opt_ind, cr)	Nil	opt:= opt_ind; S_credit:=cr; R_credit:=0; U!TCONind
t_2 $s_2 \rightarrow s_4$	N?TrCC(opt_ind, cr)	opt_ind < opt	TRsq:=0; TSsq:=0; opt:=opt_ind; S_credit:=cr; U!TCONconf
t_3 $s_2 \rightarrow s_5$	N?TrCC(opt_ind, cr)	opt_ind > opt	U!TDISind; N!TrDR
t_4 $s_2 \rightarrow s_1$	N?TrDR(disc_reason, switch)	Nil	U!TDISind; N!terminated
t_5 $s_3 \rightarrow s_4$	U?TCONresp(accpt_opt)	accpt_opt< opt	opt:= accpt_opt; TRsq:=0; TSsq:=0; N!TrCC
t_6 $s_3 \rightarrow s_6$	U?TDISreq()	Nil	N!TrDR
t_7 $s_4 \rightarrow s_4$	U?TDATAreq(Udata, E0SDU)	S_credit > 0	S_credit:= S_credit -1; TSsq := (TSsq +1)mod128; N!TrDT
t_8 $s_4 \rightarrow s_4$	N?TrDT(Send_sq, Ndata, E0TSDU)	R_credit <> 0 and Send_sq= TRsq	TRsq:=(TRsq+1)mod128; R_credit:=R_credit -1; U!DATAind; N!TrAK
t_9 $s_4 \rightarrow s_4$	N?TrDT(Send_sq, Ndata, E0TSDU)	R_credit = 0 V Send_sq <> TRsq	U!error; N!error
t_{10} $s_4 \rightarrow s_4$	U?U READY(cr)	Nil	R_credit:= R_credit + cr; N!TrAK
t_{11} $s_4 \rightarrow s_4$	N?TrAK(XpSsq, cr)	TSsq $\geq$ XpSsq **&** cr + XpSsq – TSsq $\geq$ 0 **&** cr +XpSsq – TSsq $\leq$ 15	S_credit := cr + XpSsq – TSsq
t_{12} $s_4 \rightarrow s_4$	N?TrAK(XpSsq, cr)	TSsq $\geq$ XpSsq **&** (cr + XpSsq – TSsq < 0 **V** cr +XpSsq – TSsq >0**)**	U!error; N!error
t_{13} $s_4 \rightarrow s_4$	N?TrAK(XpSsq, cr)	TSsq < XpSsq **&** cr + XpSsq – TSsq – 128 $\geq$ 0 **&** cr + XpSsq – TSsq – 128 $\leq$ 15	S_credit:= cr+ XpSsq –TSsq – 128
t_{14} $s_4 \rightarrow s_4$	N?TrAK(XpSsq, cr)	TSsq < XpSsq **&** (cr + XpSsq – TSsq – 128 < 0 **V** cr + XpSsq – TSsq – 128 > 15 **)**	U!error; N!error
t_{15} $s_4 \rightarrow s_4$	N?Ready()	S_creidit >0	U!Ready
t_{16} $s_4 \rightarrow s_5$	U?TDISreq()	Nil	N!TrDR
t_{17} $s_4 \rightarrow s_6$	N?TrDR(disc_reason, switch)	Nil	U!TDISind; N!TrDC
t_{18} $s_6 \rightarrow s_1$	N?terminated()	Nil	U!TDISconf
t_{19} $s_5 \rightarrow s_1$	N?TrDC()	Nil	N!terminated; U!TDISconf
t_{20} $s_5 \rightarrow s_1$	N?TrDR(disc_reason, switch)	Nil	N!terminated

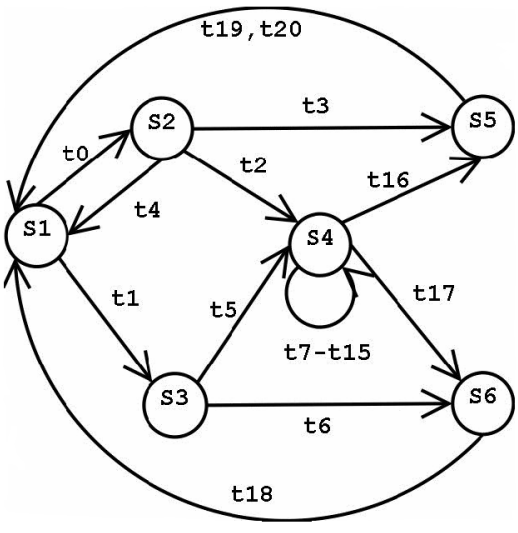

Fig. 2. Class 2 transport protocol EFSM model

experiments for research purposes. Fig. 1 shows the INRES initiator EFSM together with the transitions' specifications.

The second case study is a major model based on the *AP-module* of the simplified version of a class 2 transport protocol. The EFSM model represents the core protocol transitions as described in [16] and [24]. This EFSM has two interaction points *U* and *N* for connecting to transport service access point and a mapping module respectively. The considered EFSM is involved in connection establishment, data transfer, end-to-end flow control and segmentation.The model transitions are shown in Fig.2 and described in Table1. The class 2 transport protocol EFSM is defined by:

- $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$
- $s_0 = s_1$
- $V = \{opt, R\text{-}credit, S\text{-}Credit, TRsq , TSsq\}$
- $I = \{U?TCONconf(dst\_add, prop\_opt), N?TrCR(peer\_add, opt\_ind, cr), N?TrCC(opt\_ind, cr), N?TrDR(disc\_reason, switch), U?TCONresp(accpt\_opt), U?TDISreq, U?TDATAreq(Udata, EoSDU), N?TrDT(Send\_sq, Ndata, EoTSDU), U?UREADY(cr), N?TrAK(XpSsq, cr), N?Ready, U?TDISreq(), N?TrDC, N?terminated\}$
- $O = \{U!TCONreq(opt), U!TCONind(peer\_add, opt), U!TDISind(msg), U?TDISconf, U!TDATind(data, EoTSDU), U!error, U!READY, N!TrCR(dest\_add, opt, credit), N!TrDR(reason, switch), N!terminated, N!TrCC(opt, credit), N!TrDT(sq\_no, data, EoSDU), N!TrAK(XpSsq, cr), N!error, N!TrDC\}$
- $T = \{ t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}\}$

In these two EFSMs, all the variables in the variable set *V* and the input parameters in the input declarations set *I* are of integer data type.

2.3. Evolutionary algorithms (EAs)

Evolutionary algorithms are optimisation searching techniques that adopt the evolution notion as a search mechanism. Genetic Algorithms (GAs), probably the most common type of EA, were introduced by John Holland [26] and his colleagues in the United State in the early 1970s where they implemented the natural selection theory to present a powerful, simple, and sturdy technique which can be applied to solve optimization problems. GAs work on a set of candidate solutions, referred to as a population, rather than a single solution and this allows more points in the search domain to be sampled. GAs require a suitable representation of solutions in order to be applied to a particular problem. This can be achieved by using solution encoding. When representing solutions, each solution is called a chromosome. Each chromosome consists of many components, these components are called genes. Each gene may have different possible values. The value, that a gene may have, is called an allele. The locus refers to a gene's position in the chromosome string [27]. GAs work on genotypes which refer to the encoded structures of solutions whereas the decoded structures of solutions are called phenotype. For example, let the initial set of solutions be integer values such as {7, 6, 8}. Each number in this set is called phenotype while the following set {0111, 0110, 1000} represents chromosomes or genotypes. Any bit within the above strings represents a gene with either 0 or 1 value which is an allele.

The GAs cycle starts by evaluating the fitness of each individual. The fitness of an individual is a positive value that measures how 'fit' this individual and hence its chance of use as a parent.

Then a selection based on fitness is made to perform 'breeding'. There are many selection methods that can be used e.g., Roulette wheel and ranking selections, however, the question about which method is better is problem dependent [28].

Through 'breeding' new individuals are introduced. This is accomplished by applying a crossover operator (also called recombination). Crossover acts on two individuals to produce two new individuals and can be performed in several ways. The simplest one is referred to as one-point crossover. It operates by choosing a random position on the chromosome's bit string, and then the substrings before that position are kept while the tails are swapped [29]. For example, if the two parents' chromosomes are P_1 and P_2 with crossing point at locus 4, then C_1 and C_2 are the offspring chromosomes.

$$\begin{matrix} P_1\ \{011|00\} \\ P_2\ \{101|11\} \end{matrix} \Longrightarrow \begin{matrix} C_1\ \{011|11\} \\ C_2\ \{011|00\} \end{matrix}$$

In order to maintain population diversity, new characteristics are infrequently injected by applying mutation. Mutation acts on one chromosome at a time, where it randomly changes the values of some of the chromosome's genes [29]. For example, the chromosomes C_1 above might become C_1' after mutating the genes on locus 1 and 5.

$$C_1\ \{01111\} \Longrightarrow C_1'\ \{11110\}$$

The GAs cycle will most probably yield 'fitter' individuals referred to as a new generation and these are used to update the population. The population undergoes a number of updates until fulfilling one of the stopping criteria such as finding the best solution or reaching a maximum number of generations [30].

2.4. Evolutionary testing (ET)

Evolutionary testing (ET) is a technique that employs EA to automatically generate test data from a test target. In test data generation, EA is employed to solve a minimization problem where the lower the fitness of a solution the better it is and the optimal solution(s) will have a fitness equal to zero. Two aspects are central in applying ET to generate test data automatically. First, the test adequacy criterion which is a property that a test must satisfy in order to be considered sufficient. Many test adequacy criteria require that a set of structures in the code or specification to be covered in testing [4]. For example, we might require that all of the statements in the code are exercised (covered) in testing (statement coverage). The second is the objective function which will be used to evaluate the members of the population. If we considered the branch coverage test adequacy criterion, then all the branches in the subject program need to be taken (covered). In this case, an objective function that depends on a branch distance can be used in order to evaluate the input values. A branch distance is computed to measure how close a particular input was to executing the target branch that is missed e.g., $|A\text{-}B|$ is the branch distance for the predicate ($A > B$). The lower $|A\text{-}B|$ is the closer is A to B and the closer the test is to taking the branch. A full list of different types of conditions and their branch distance computations is provided by Tracey et al. [31].

Often, programs have nested predicates, for example an *IF* statement could be contained in a loop. In this case, an objective function which only employs branch distance is not sufficient and extra information is needed to guide the search. This is given in terms of approach level or

approximation level [32] which measures how close an input vector was to executing the structure under test. A central notion to approach level calculation is a *critical node* which is a branching node at which the path control flow may divert. An approach level is calculated by subtracting 1 from the number of the critical nodes away from the target node (Equation 2). For example, Fig. 5 (Node A) shows the approach level calculation for 4 critical nodes away from *Subtarget-1*. If the *false* branch of any of them (shown in dashed arrows) is taken then the target node is missed. Since it is necessary that the branch distance of the upper IF statement is always greater than the ones in a lower level, the branch distance of each IF statement is normalized to a value in the range of [0..1] (Equation 1). The normalized branch distance is then added to the associated approach level of that branch to form the fitness value of that branch (Equation 3). In this way, the cost associated with each branch is clearly contrasted among other and so the branch distance of the upper *IF* statement is always greater than the branch which is located in a lower level. Therefore, a set of test data that achieve more conditions (longer path) is always associated with better (lower) fitness that that which achieve fewer conditions.

A recent survey [2] has focused on evolutionary test data generation. The technique presented in this paper adapts the notion of branch distance and approach level in order to construct a new fitness function, described in Section 3, for an automatic generation of a path test data through EFSM models.

$$norm\ (d) = 1 - 1.05^{-d} \quad (1)$$

$$approach\ level = numOfCriticalNodesAwayFromTarget - 1 \quad (2)$$

$$fitness = approach\ level + norm\ (d) \quad (3)$$

Where d is a branch distance, and norm (d) is the branch distance value scaled between [0, 1].

2.5. Related work

Many test generation approaches for systems modeled as EFSMs appear in the literature [10], [11], [12], [13], [15], [16], [18], [20], [22], [33] and [34]. An approach to generate a unified test sequence (UTS) for EFSM models is presented in [12] based on two techniques: one to test the control part (FSM) and the other to test the data part by using data flow analysis technique. The resultant UTS is then checked for executability by using a *constraint satisfaction* method. However, some assumptions about the EFSM model i.e. the existence of *self-loop influencing* (a loop that modifies a global predicate variable) may not be applicable for other EFSM models.

Generating test sequence for EFSM models by employing functional program testing is studied in [20]. The approach converts the specification written in Estelle [35] into a simpler form in order to construct control and data flow graphs to be used in test sequence derivation. However, the approach does not allow automatic generation of the test sequences. Also, the approach does not allow some common code constructs such as functions calls and conditional statements.

Other methods that test an EFSM model using FSM-based test techniques appear in [15], [33] and [34]. A technique that transforms a class of EFSM models into a class of FSMs is used by [33] for the purpose of deriving test sequences from VHDL or bestmap-C hardware specifications. The transformation allows available FSM testing methods to be applied to each resultant FSM separately. This technique requires that all the input variables have finite domains. However, the approach may easily lead to a large number of configurations which cannot be

handled in practice. A detailed study of various FSM-based test generation techniques for the purpose of fault coverage was conducted by [15], whereas a study of the four main formal methods *paths, distinguishing sequences, characterizing sequences and unique I/O (UIO) sequences* for protocol testing based on FSM models is presented in [34].

Generally, the notion of testing EFSM models based on FSM methods requires a transformation from EFSM to FSM model. There are two approaches to conduct this: the first is to abstract the data from an EFSM model so the resultant is an FSM model. The limitation of this approach is that the paths taken from the FSM model are not necessarily feasible in the corresponding EFSM model. The second approach is to expand an EFSM model to become an FSM model, however, the number of states in the resultant FSM can easily become prohibitively large [36].

A technique for generating unique state identification sequences for EFSM models is presented in [16, 18]. The technique is based on computing a new type of state identification for each state called *context independent unique sequence (CIUS)*. This requires that all the paths that start from any state be context independent. That is, all the guards included in any path can be interpreted symbolically. Furthermore, each state must have a CIUS. Since these two requirements may not be satisfied by a given EFSM model, the applicability of this approach is limited. Furthermore, the approach does not provide a method to generate input test data to be used in testing the generated paths.

An approach which employs software data flow testing to derive a test sequence from EFSM models is presented in [22]. The selection of each test case depends on identifying all the associations between each output and all the inputs that affects that output. A potential limitation of this approach is the cost associated with the analysis phase when nontrivial EFSM models are considered.

A test sequence generation approach that overcomes the feasibility problem in advance is introduced by [10, 11]. The approach requires that all the conditions and actions in an EFSM model are linear, functions or procedures should be transformed, and the target model represents a single process. The approach converts a class of EFSMs into consistent EFSMs in order to enable test sequences generation. However, the approach does not provide a technique to generate input test sequence that will test the generated paths.

Another study that focuses on the EFSM path feasibility problem is presented in [13]. The study presents a technique to bypass the infeasible path problem in an EFSM model through two steps. First, the *SDL* (*Specification and Description Language*) model specifications are rewritten in order to derive a *normal form-EFSM* (*NF-EFSM*). Second, the resultant *NF-EFSM* is extended to *Expanded-EFSM* (*EEFSM*) in order to aid the testability. As a result, all the paths presented in the output *EEFSM* are feasible. However, the study does not tackle the problem of generating test data for the final output feasible paths.

A recent study about a fitness calculation method in the presence of function calls appears in [37, 38]. The study proposed a fitness calculation method to derive test data from a state machine. The approach of fitness calculation presented in Tracey et al. [31] is first applied to each function, and then path fitness is constructed by considering each function in the path as a critical node where objective function value is calculated by adding each function's return value to its associated approach level in the considered path. Experiments were conducted on a set of

Java classes and showed that the proposed approach was successful in producing valid test input. However, the limitation of this study is the assumption that each function does not have an internal path i.e. nested *IF* statements for which Tracey et al. [31] approach does not always provide a sufficient guidance as argued in [2, 32].

Another study that describes a fitness calculation in the presence of flow functions calls appears in [39]. The study compares the conventional fitness calculation approach and a proposed fitness calculation in the presence of functions calls. The experiment was conducted on two case studies and showed that the proposed fitness calculation was faster and successfully achieved the test targets in all the tries while the conventional fitness calculation was slower and failed in some tries. However, the study assumes that a set of functions are manipulating the same input parameters i.e. each function receives the same input values received by the previously called function which is a special case of functions calls.

3. Proposed test generation approach

Although the available techniques highlighted in Section 2, made considerable contributions towards the domain of EFSM testing, the problem of generating test data to follow a given feasible path in an EFSM has received little attention. The motivation of our study is that all present techniques [5], [9-13] and [16-21] that produce tests from an EFSM model through generating a set of paths can potentially incorporate our technique.

The fitness calculation method described previously in Subsection 2.4 is effective in structural testing where the test target is represented as a single node in the main body of the function or the program. However, this technique is not designed to cope with the case when there is a test objective that involves calls to a set of transitions sequentially. In this case, the main test target comprises a set of subtargets that have to be achieved in order to achieve the main test target i.e. taking successfully the last transition in a path.

For example, in functional testing it is necessary to trigger a path in order to reach a specific state in the machine. In this scenario, the first transition in the path must be triggered successfully in order to be able to try to trigger the next transition and so forth. Since an EFSM transition can be considered as a function with input parameters and conditions, the problem of generating test data to trigger a given path can be seen as finding suitable input parameter values to be applied to each transition (function) in that path in a sequential way.

In order to describe the proposed fitness calculation method, consider the first function (fun_1) shown in Fig.3 which requires two suitable input values to achieve a set of four nested *IF* statements. For a given path comprising the transition sequence $fun_1(x_1,y_1) \rightarrow fun_1(x_2,y_2) \rightarrow fun_1(x_3,y_3)$, the search should first locate suitable input values (x_1,y_1) that trigger successfully the first transition before it can progress to find the next suitable input values (x_2,y_2) that trigger the next transition in the path and so forth.

The manipulation of a path in this way is similar to the structure of nested *IF* statements where each *IF* statement compares the associated function's return value with 0. If a function is successfully triggered then its return value is set to 0 otherwise it should reflect the fitness of the input values in respect only to this particular function (we will refer to this fitness value henceforth as the *function distance*). In this way, the first transition in the path can be considered as the upper *IF* statement and then functions which come next are treated as nested *IF*

```
Double fun1( int x, int y)
 {
  if x >=10
   {if x <=20
    {if y >=0
     {if y <=10
      //result = 0 //Target achieved
 }}}}
```

```
Double fun2( int x, int y, int z)
 {
  if x >=10
   {if x <=20
    {if y >=0
     {if y <=10
      {if z >=30
       {if z <= 40
        //result = 0 //Target achieved
 }}}}}}
```

Fig.3 Two function case studies with nested *IF* statements

statements. Therefore, the fitness function for a given path can be based upon the work of Wegener et al. [32] where the approach level here represents *transition approach level* which measures how close a given set of input was to taking the target (final) transition (see Equation 4). The transition approach level is derived in the same way as the approach level described previously. That is, for a given path, any transition which has guard(s) is considered a critical transition and so the transition approach level is derived by subtracting 1 from the number of critical transitions away from the target transition (Equation 5).

The value of *function distance* will determine whether or not the corresponding function is achieved. The previous work of [37, 38] calculated this component by using the approach of Tracey et al. [31] which is effective when conditions within each function are linked by '*AND*' operator. Nevertheless, in the presence of nested *IF* statements, the use of Tracey et al. [31] approach leads to some plateaux in the corresponding objective function landscape which is not the case if Wegner et al. [32] approach is applied instead. For example, Fig.4 shows the objective function landscapes of the first function case study (fun_1) given in Fig.3. The unnecessary plateaux of the landscape of Tracey et al. [31] approach can cause the search to be hindered or even falls in a local minima. A similar case study that compares Tracey et al. [31] and Wegener et al. [32] approaches is reported in [2]. Therefore, Applying the approach of Wegener et al. [32] to calculate a *function distance* is more efficient and provide a better guidance (Equation 6).

$$path\ fitness = norm\ (function\ distance) + transition\ approach\ level \quad (4)$$

$$transition\ approach\ level = NumOfCrticalTransAwayFromTarget - 1 \quad (5)$$

$$function\ distance = norm(branch\ distance) + approach\ level \quad (6)$$

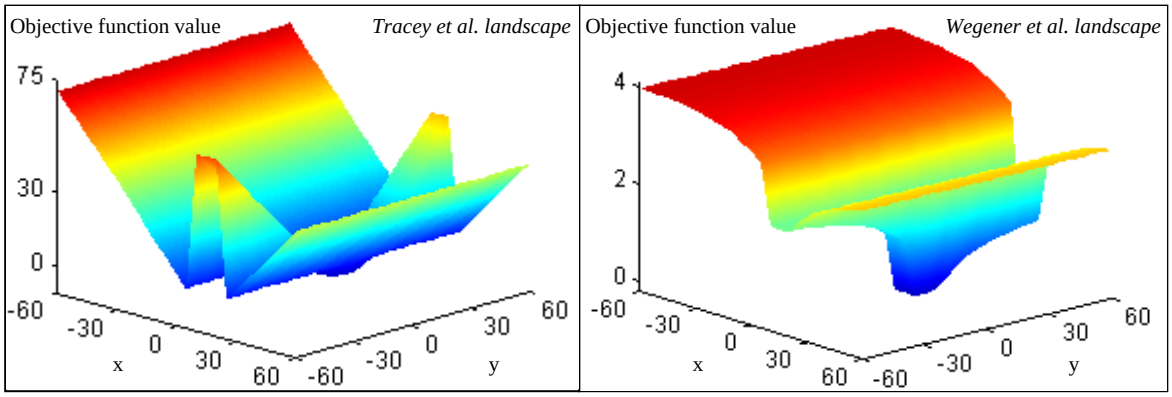

Fig.4 Objective function landscapes of the first function case study by using Teracy et al. and Wegener et al. fitness calculations.

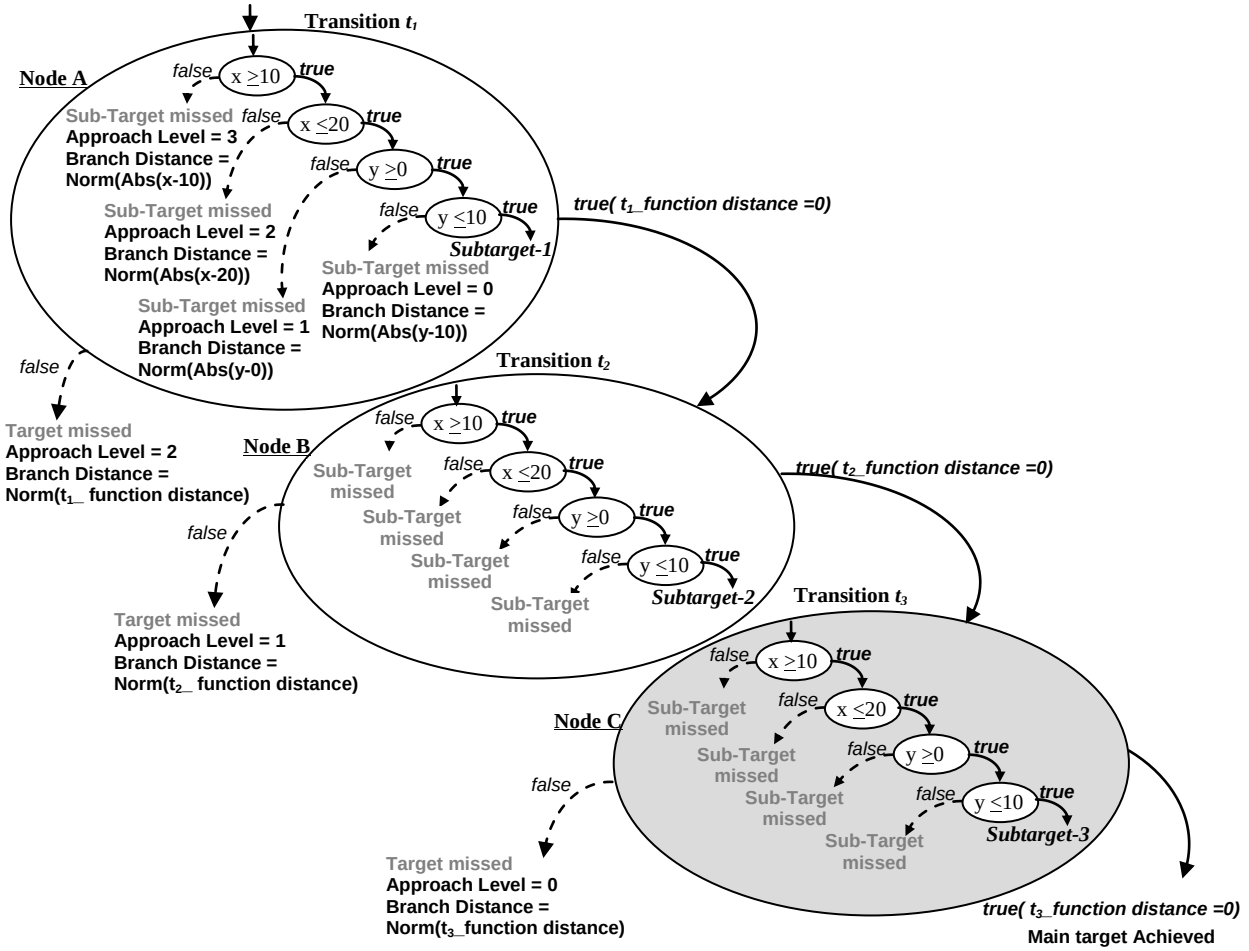

Fig.5 The proposed fitness calculation method applied to a path case study which consists of three sequential calls to function fun_1.

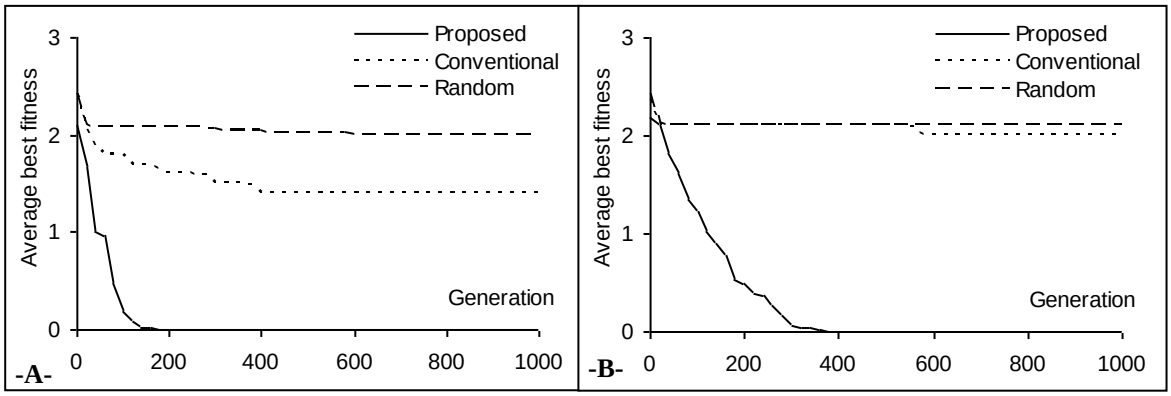

Fig.6 Two fitness plots that compare between the conventional and proposed approaches on the two path case studies. Plot -A- is for $path_1$ and plot -B- is for $path_2$.

The proposed method of calculating a path fitness can be seen simply as applying the approach of Wegener et al. [32] two times: the first is to calculate a *function distance* for each function in a path whereas the next time is to calculate a path fitness after considering each function as a nested *IF* statement. For example, Fig. 5 shows the fitness calculation for the path

$fun_1(x_1,y_1) \rightarrow fun_1(x_2,y_2) \rightarrow fun_1(x_3,y_3)$ using the proposed method. However, the alternative option is to apply Tracey et al. [31] method in order to calculate a *function distance* for each function in a path and then use Wegener et al. [32] method to calculate the path fitness. We will refer to this alternative option as the *conventional* approach for the purpose of justifying the proposed method. The conventional approach is the same as the approach reported in [37, 38] which is the only previous work that utilize GAs for testing form state machines.

Naturally, transitions' guards are either sequenced as nested *IF* statements or linked with logical operators *'AND'* and *'OR'*, for such a case, in order to apply the proposed fitness calculation method, transition's guards linked with *'AND'* operator will be represented as nested *IF* statements when calculating a *function distance*. It is always possible to convert conditions linked with *'AND'* to nested *IF* statements, however the reverse is not always valid. Thus, the proposed approach has an advantage over the conventional one since it can deal with both cases.

If a transition guards are linked with *'OR'* operator, we split the transition into a number of transitions equal to the number of '*OR*' operators used + 1. One of the benefits is that we test each condition and so no guards are left unexercised, however, an alternative method would be to apply an approach similar to that of Tracey et al. [31] where the minimum fitness value for a set of conditions linked with *'OR'* operator is used.

In order to apply an ET- based technique, a suitable form of solution encoding is required. This can be selected according to the considered machine input parameters type. That is, it is possible to use *binary* or *integer* encoding when all of the considered machine input parameters are of *integer* data type; however, if some of the machine input parameters are of *double* data type then real valued encoding can be used. A candidate solution that represents a path test data comprises a set of components where each component represents one input parameter. For example, a possible solution encoding of the path case study shown in Fig.5 consists of six components of type *integer* $<C_0, C_1, C_2, C_3, C_4, C_5>$.

3.1. Fitness function justification

In order to justify the proposed path fitness calculation, we present two path case studies: the first case study is $path_1$: $fun_1(x_1,y_1) \rightarrow fun_1(x_2,y_2) \rightarrow fun_1(x_3,y_3)$ shown in Fig.5 whereas the second case study consists of three sequential calls to the function (fun_2) given in Fig.3 $path_2$: $fun_2(x_1,y_1,z_1) \rightarrow fun_2(x_2,y_2,z_2) \rightarrow fun_2(x_3,y_3,z_3)$. The differences between these two paths are the number of required input parameters and the level of nested *IF* statements in each function. In order to estimate the ease with which each path can be triggered, we applied a random algorithm to each path. Also we applied the conventional and the proposed approaches for the purpose of comparison. In this experiment, a population of 100 individuals and integer valued encoding was used with a range of [-1000..1000] for each input parameters. The experiment was conducted 10 times and each time 1000 generations were allowed before the search was terminated. The graphs plotted in Fig.6 (-A- for $path_1$ and B- for $path_2$) show the best fitness obtained so far in the search for a particular generation, averaged over ten repetitions of the experiment. For both paths, it was not surprising that the random search failed in finding the required input values and so these two paths cannot be easily triggered.

For $path_1$, due to the almost unidirectional landscape of the conventional approach, this approach failed to make any progress after 400 generations. Nevertheless, the landscape of the proposed approach provided better guidance and it hit the target after 259 generations. For $path_2$,

the landscape of the conventional approach was worse than that observed on $path_1$ and the performance was similar to that of the random search. This was not the case with the proposed approach where its landscape was similar to that observed on $path_1$ and provided better guidance to hitting the target after 440 generations. The results of this experiment suggest that the performance of the conventional approach can be inefficient when nested *IF* statements are existed. It also seems likely that the performance of the conventional approach will be even worse when longer paths are used. As shown in Fig.3, having more functions in a path leads to more plateaux in the final objective function landscape of that path. Therefore, there are more chances that the search may stuck in a local minima. However, this is not the case with the proposed approach since each included function landscape does not suffer from the plateaux existence in the first place.

In addition to this experiment, we have conducted another set of initial experiments with different level of nested *IF* statements and we observed that when there are 3 or fewer nested *IF* statements in each function of a path, the conventional approach performs almost similarly to the proposed one and this is because fewer plateaux exist in each function landscape.

4. Empirical verification

4.1. Experiment design

In designing our experiment, we aimed to evaluate the efficiency of the proposed fitness function in guiding the search for test data that can trigger the subject transition paths (TPs) under the test. In order to achieve this, there are three factors to be considered.

The first is related to the length of TPs to be generated. Naturally, a short TP is likely to be easy to trigger since it has fewer guards. In order to avoid the impact of this factor in our experiment, we want to generate TPs that are relatively long and cover various TPs lengths.

The second factor is related to the number of input parameters required to trigger a given subject TP. A TP that requires fewer input parameters is typically easier to trigger. For example, finding ten parameter values to be applied to a given TP transitions is usually harder than finding one parameter value. As a result, we generated TPs that require relatively many input parameters. We therefore eliminate any generated TP that requires less than 4 input parameters.

The third factor is to determine how easy it is to trigger a generated TP; we use a random test data generator to assess this. For example, if we can quickly randomly find the suitable input values to trigger a generated TP then we can state that this TP is easy to trigger.

Since the subject EFSM that represents INRES protocol (see Fig.1) consists of 15 transitions, we randomly[1] generated two sets of 15 subject TPs (see Table 2 and Table 3) with a length of 8 and 11 transitions respectively. In these two sets, each TP is responsible for exercising a particular transition and so both sets define, if all TPs are feasible, a transition coverage test suite for INRES initiator EFSM. Also, the subject TPs in these two sets required various number of input parameters and so they include a variety of TP cases.

For the subject EFSM that represents the class 2 transport protocol, which consists of 21 transitions (see Fig.2), we randomly generated three sets of 21 TPs with each set representing a transition coverage test suite for the considered EFSM. The lengths of the subjects TPs in each

[1] We use a random path generator for the purpose of evaluating our proposed approach; however, there are many other efficient approaches to generate a set of paths through an EFSM model that includes all the machine's transitions.

Table 2. The first set of subject TPs for INRES protocol: each TP consists of 8 transitions

Path ID	Params	Subject paths
$I_{1\text{-}0}$	6	$\mathbf{T_0}$(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→$T_6(P_2)$→$T_4(P_3)$ →$T_5(P_4)$→$T_4(P_5)$
$I_{1\text{-}1}$	6	T_0(none)→$\mathbf{T_1}$(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_5(P_4)$→$T_4(P_5)$
$I_{1\text{-}2}$	5	T_0(none)→$\mathbf{T_2}$(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→$T_6(P_2)$→$T_4(P_3)$→$T_5(P_4)$
$I_{1\text{-}3}$	4	T_0(none)→$\mathbf{T_3}$(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$
$I_{1\text{-}4}$	6	T_0(none)→T_1(none)→$\mathbf{T_4}(P_0)$→$T_7(P_1)$→$T_7(P_2)$→$T_6(P_3)$→$T_4(P_4)$→$T_5(P_5)$
$I_{1\text{-}5}$	6	T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$\mathbf{T_5}(P_3)$→$T_4(P_4)$→$T_6(P_5)$
$I_{1\text{-}6}$	6	T_0(none)→T_1(none)→$T_4(P_0)$→$\mathbf{T_6}(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_7(P_4)$→$T_5(P_5)$
$I_{1\text{-}7}$	6	T_0(none)→T_1(none)→$T_4(P_0)$→$\mathbf{T_7}(P_1)$→$T_6(P_2)$→$T_4(P_3)$→$T_7(P_4)$→$T_5(P_5)$
$I_{1\text{-}8}$	5	T_0(none)→T_2(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→$T_7(P_2)$→$T_7(P_3)$→$\mathbf{T_8}(P_4)$
$I_{1\text{-}9}$	5	T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_5(P_3)$→$T_4(P_4)$→$\mathbf{T_9}$(none)
$I_{1\text{-}10}$	5	T_0(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→$T_7(P_2)$→$T_7(P_3)$→$T_7(P_4)$→$\mathbf{T_{10}}$(none)
$I_{1\text{-}11}$	5	$\mathbf{T_{11}}$(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_5(P_3)$→$T_4(P_4)$
$I_{1\text{-}12}$	4	T_0(none)→$\mathbf{T_{12}}$(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→$T_7(P_2)$→$T_7(P_3)$
$I_{1\text{-}13}$	5	T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_5(P_4)$→$\mathbf{T_{13}}$(none)
$I_{1\text{-}14}$	5	T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_5(P_3)$→$T_4(P_4)$→$\mathbf{T_{14}}$(none)

Table 3. The second set of subject TPs for INRES protocol: each TP consists of 11 transitions

Path ID	Params	Subject paths
$I_{2\text{-}0}$	5	T_{11}(none)→T_{11}(none)→$\mathbf{T_0}$(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→$T_6(P_2)$→$T_4(P_3)$→T_9(none)→$T_5(P_4)$ →T_{13}(none)
$I_{2\text{-}1}$	6	T_0(none)→$\mathbf{T_1}$(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_5(P_3)$→$T_4(P_4)$→T_9(none)→$T_7(P_5)$→T_{14}(none)→T_0(none)
$I_{2\text{-}2}$	8	T_0(none)→$\mathbf{T_2}$(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_5(P_3)$→$T_4(P_4)$→$T_7(P_5)$→$T_6(P_6)$→$T_4(P_7)$
$I_{2\text{-}3}$	4	T_0(none)→T_2(none)→T_2(none)→$\mathbf{T_3}$(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_5(P_3)$
$I_{2\text{-}4}$	5	T_0(none)→T_1(none)→$\mathbf{T_4}(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_5(P_4)$→T_{13}(none)→T_0(none)→T_2(none)→T_{12}(none)
$I_{2\text{-}5}$	7	T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→T_9(none)→$T_7(P_3)$→T_9(none)→$\mathbf{T_5}(P_4)$→$T_4(P_5)$→$T_6(P_6)$
$I_{2\text{-}6}$	5	T_0(none)→T_{12}(none)→T_{11}(none)→T_0(none)→T_2(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→$\mathbf{T_6}(P_2)$→$T_4(P_3)$→$T_5(P_4)$
$I_{2\text{-}7}$	6	T_0(none)→T_{12}(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→T_9(none)→$\mathbf{T_7}(P_3)$→$T_5(P_4)$→$T_4(P_5)$
$I_{2\text{-}8}$	7	T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_7(P_4)$→$T_7(P_5)$→T_9(none)→$\mathbf{T_8}(P_6)$→T_0(none)
$I_{2\text{-}9}$	7	T_{11}(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$\mathbf{T_9}$(none)→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_5(P_4)$→$T_4(P_5)$→$T_6(P_6)$
$I_{2\text{-}10}$	5	T_0(none)→T_{12}(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→T_9(none)→$T_7(P_3)$→$T_7(P_4)$→$\mathbf{T_{10}}$(none)
$I_{2\text{-}11}$	7	$\mathbf{T_{11}}$(none)→T_0(none)→T_2(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_5(P_4)$→$T_4(P_5)$→$T_6(P_6)$
$I_{2\text{-}12}$	6	T_0(none)→$\mathbf{T_{12}}$(none)→T_{11}(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_7(P_3)$→$T_7(P_4)$→$T_5(P_5)$
$I_{2\text{-}13}$	5	T_{11}(none)→T_0(none)→T_1(none)→$T_4(P_0)$→$T_7(P_1)$→T_9(none)→$T_6(P_2)$→$T_4(P_3)$→$T_5(P_4)$→$\mathbf{T_{13}}$(none)→T_{11}(none)
$I_{2\text{-}14}$	6	T_0(none)→T_1(none)→$T_4(P_0)$→$T_6(P_1)$→$T_4(P_2)$→$T_5(P_3)$→$T_4(P_4)$→$T_7(P_5)$→T_9(none)→$\mathbf{T_{14}}$(none)→T_{11}(none)

Table 4. The first set of subject TPs for the class 2 transport protocol: each TP consists of 5 transitions

Path ID	Params	Subject paths
$P_{1\text{-}0}$	6	$\mathbf{T_0}(P_0)$→T_4(none)→$T_1(P_1,P_2)$→$T_5(P_3)$→$T_{11}(P_4,P_5)$
$P_{1\text{-}1}$	6	$\mathbf{T_1}(P_0)$→$T_5(P_1)$→$T_{13}(P_2,P_3)$→$T_{11}(P_4,P_5)$→T_{16}(none)
$P_{1\text{-}2}$	8	$T_0(P_0)$→$\mathbf{T_2}(P_1,P_2)$→$T_{11}(P_3,P_4)$→$T_{11}(P_5,P_6)$→$T_9(P_7)$
$P_{1\text{-}3}$	6	$T_0(P_0)$→$\mathbf{T_3}(P_1,P_2)$→T_{19}(none)→$T_0(P_3)$→$T_3(P_4,P_5)$
$P_{1\text{-}4}$	6	$T_0(P_0)$→$\mathbf{T_4}$(none)→$T_0(P_1)$→$T_2(P_2,P_3)$ →$T_{13}(P_4,P_5)$
$P_{1\text{-}5}$	8	$T_1(P_0,P_1)$→$\mathbf{T_5}(P_2)$→$T_{13}(P_3,P_4)$→$T_{12}(P_5,P_6)$→$T_9(P_7)$
$P_{1\text{-}6}$	5	$T_0(P_0)$→$T_3(P_1,P_2)$→T_{20}(none)→$T_1(P_3,P_4)$→$\mathbf{T_6}$(none)
$P_{1\text{-}7}$	7	$T_1(P_0,P_1)$→$T_5(P_2)$→$T_{11}(P_3,P_4)$→$\mathbf{T_7}$(none)→$T_{13}(P_5,P_6)$
$P_{1\text{-}8}$	6	$T_0(P_0)$→$T_2(P_1,P_2)$→$T_{10}(P_3)$→$\mathbf{T_8}(P_4)$→$T_9(P_5)$
$P_{1\text{-}9}$	6	$T_1(P_0,P_1)$→T_5(,P_2)→$T_{12}(P_3,P_4)$→$\mathbf{T_9}(P_5)$→T_{15}(none)
$P_{1\text{-}10}$	8	$T_0(P_0)$→$T_2(P_1,P_2)$→$T_{11}(P_3,P_4)$→$\mathbf{T_{10}}(P_5)$→$T_{11}(P_6,P_7)$
$P_{1\text{-}11}$	7	$T_1(P_0,P_1)$→$T_5(P_2)$→$\mathbf{T_{11}}(P_3,P_4)$→$T_{13}(P_5,P_6)$→T_{16}(none)
$P_{1\text{-}12}$	7	$T_1(P_0,P_1)$→$T_5(P_2)$→T_{15}(none)→$T_{13}(P_3,P_4)$→$\mathbf{T_{12}}(P_5,P_6)$
$P_{1\text{-}13}$	7	$T_1(P_0,P_1)$→$T_5(P_2)$→$\mathbf{T_{13}}(P_3,P_4)$→$T_{13}(P_5,P_6)$→T_{17}(none)
$P_{1\text{-}14}$	9	$T_0(P_0)$→$T_2(P_1,P_2)$→$T_{11}(P_3,P_4)$→$T_{12}(P_5,P_6)$→$\mathbf{T_{14}}(P_7,P_8)$
$P_{1\text{-}15}$	5	$T_0(P_0)$→$T_2(P_1,P_2)$→$T_{10}(P_3)$→$T_8(P_4)$ →$\mathbf{T_{15}}$(none)
$P_{1\text{-}16}$	7	$T_0(P_0)$→$T_2(P_1,P_2)$→$T_{14}(P_3,P_4)$→$T_{12}(P_5,P_6)$→$\mathbf{T_{16}}$(none)
$P_{1\text{-}17}$	5	$T_1(P_0,P_1)$→$T_5(P_2)$→$T_{11}(P_3,P_4)$→$\mathbf{T_{17}}$(none)→T_{18}(none)
$P_{1\text{-}18}$	5	$T_1(P_0,P_1)$→T_6(none)→$\mathbf{T_{18}}$(none)→$T_1(P_2,P_3)$→$T_5(P_4)$
$P_{1\text{-}19}$	6	$T_0(P_0)$→$T_3(P_1,P_2)$→$\mathbf{T_{19}}$(none)→$T_0(P_3)$→$T_3(P_4,P_5)$
$P_{1\text{-}20}$	6	$T_0(P_0)$→$T_3(P_1,P_2)$→$\mathbf{T_{20}}$(none)→$T_0(P_3)$→$T_3(P_4,P_5)$

Table 5. The second set of subject TPs for the class 2 transport protocol: each TP consists of 8 transitions

Path ID	Params	Subject paths
$P_{2\text{-}0}$	11	$\mathbf{T_0}(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{14}(P_3,P_4)\rightarrow T_{14}(P_5,P_6)\rightarrow T_{13}(P_7,P_8)\rightarrow T_{12}(P_9,P_{10})\rightarrow T_{16}(\text{none})\rightarrow T_{20}(\text{none})$
$P_{2\text{-}1}$	6	$\mathbf{T_1}(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_{10}(P_3)\rightarrow T_{15}(\text{none})\rightarrow T_{15}(\text{none})\rightarrow T_{14}(P_4,P_5)\rightarrow T_{16}(\text{none})\rightarrow T_{20}(\text{none})$
$P_{2\text{-}2}$	9	$T_0(P_0)\rightarrow T_3(P_1,P_2)\rightarrow T_{20}(\text{none})\rightarrow T_0(P_3)\rightarrow \mathbf{T_2}(P_4,P_5)\rightarrow T_9(P_6)\rightarrow T_7(\text{none})\rightarrow T_{11}(P_7,P_8)$
$P_{2\text{-}3}$	9	$T_0(P_0)\rightarrow \mathbf{T_3}(P_1,P_2)\rightarrow T_{20}(\text{none})\rightarrow T_0(P_3)\rightarrow T_4(\text{none})\rightarrow T_1(P_4,P_5)\rightarrow T_5(P_6)\rightarrow T_{13}(P_7,P_8)$
$P_{2\text{-}4}$	7	$T_0(P_0)\rightarrow \mathbf{T_4}(\text{none})\rightarrow T_0(P_1)\rightarrow T_2(P_2,P_3)\rightarrow T_9(P_4)\rightarrow T_{12}(P_5,P_6)\rightarrow T_7(\text{none})\rightarrow T_{17}(\text{none})$
$P_{2\text{-}5}$	11	$T_1(P_0,P_1)\rightarrow \mathbf{T_5}(P_2)\rightarrow T_{11}(P_3,P_4)\rightarrow T_{13}(P_5,P_6)\rightarrow T_9(P_7)\rightarrow T_7(\text{none})\rightarrow T_{13}(P_8,P_9)\rightarrow T_{10}(P_{10})$
$P_{2\text{-}6}$	9	$T_1(P_0,P_1)\rightarrow \mathbf{T_6}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_2,P_3)\rightarrow T_5(P_4)\rightarrow T_{12}(P_5,P_6)\rightarrow T_{13}(P_7,P_8)\rightarrow T_7(\text{none})$
$P_{2\text{-}7}$	10	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_9(P_3)\rightarrow T_{11}(P_4,P_5)\rightarrow \mathbf{T_7}(\text{none})\rightarrow T_{10}(P_6)\rightarrow T_9(P_7)\rightarrow T_{12}(P_8,P_9)$
$P_{2\text{-}8}$	9	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_9(P_3)\rightarrow T_7(\text{none})\rightarrow T_{10}(P_4)\rightarrow \mathbf{T_8}(P_5)\rightarrow T_9(P_6)\rightarrow T_{11}(P_7,P_8)$
$P_{2\text{-}9}$	9	$T_0(P_0)\rightarrow T_4(\text{none})\rightarrow T_1(P_1,P_2)\rightarrow T_5(P_3)\rightarrow T_{11}(P_4,P_5)\rightarrow T_{11}(P_6,P_7)\rightarrow \mathbf{T_9}(P_8)\rightarrow T_7(\text{none})$
$P_{2\text{-}10}$	11	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_{13}(P_3,P_4)\rightarrow T_{11}(P_5,P_6)\rightarrow \mathbf{T_{10}}(P_7)\rightarrow T_{15}(\text{none})\rightarrow T_{10}(P_8)\rightarrow T_{13}(P_9,P_{10})$
$P_{2\text{-}11}$	7	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{15}(\text{none})\rightarrow T_7(\text{none})\rightarrow \mathbf{T_{11}}(P_3,P_4)\rightarrow T_{14}(P_5,P_6)\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})$
$P_{2\text{-}12}$	7	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_7(\text{none})\rightarrow T_7(\text{none})\rightarrow \mathbf{T_{12}}(P_3,P_4)\rightarrow T_{13}(P_5,P_6)\rightarrow T_7(\text{none})\rightarrow T_7(\text{none})$
$P_{2\text{-}13}$	10	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{12}(P_3,P_4)\rightarrow \mathbf{T_{13}}(P_5,P_6)\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_0(P_7)\rightarrow T_3(P_8,P_9)$
$P_{2\text{-}14}$	9	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow \mathbf{T_{14}}(P_3,P_4)\rightarrow T_{14}(P_5,P_6)\rightarrow T_7(\text{none})\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_7,P_8)$
$P_{2\text{-}15}$	10	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_9(P_3)\rightarrow T_{10}(P_4)\rightarrow T_{14}(P_5,P_6)\rightarrow \mathbf{T_{15}}(\text{none})\rightarrow T_{11}(P_7,P_8)\rightarrow T_{10}(P_9)$
$P_{2\text{-}16}$	7	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow \mathbf{T_{16}}(\text{none})\rightarrow T_{19}(\text{none})\rightarrow T_1(P_3,P_4)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_5,P_6)$
$P_{2\text{-}17}$	7	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_7(\text{none})\rightarrow T_{14}(P_3,P_4)\rightarrow T_7(\text{none})\rightarrow \mathbf{T_{17}}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_5,P_6)$
$P_{2\text{-}18}$	10	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{14}(P_3,P_4)\rightarrow T_{11}(P_5,P_6)\rightarrow T_9(P_7)\rightarrow T_{17}(\text{none})\rightarrow \mathbf{T_{18}}(\text{none})\rightarrow T_1(P_8,P_9)$
$P_{2\text{-}19}$	11	$T_0(P_0)\rightarrow T_3(P_1,P_2)\rightarrow \mathbf{T_{19}}(\text{none})\rightarrow T_1(P_3,P_4)\rightarrow T_5(P_5)\rightarrow T_{13}(P_6,P_7)\rightarrow T_{10}(P_8)\rightarrow T_{12}(P_9,P_{10})$
$P_{2\text{-}20}$	8	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_{13}(P_3,P_4)\rightarrow T_{16}(\text{none})\rightarrow T_{19}(\text{none})\rightarrow T_0(P_5)\rightarrow T_3(P_6,P_7)\rightarrow \mathbf{T_{20}}(\text{none})$

Table 6. The third set of subject TPs for the class 2 transport protocol: each TP consists of 11 transitions

Path ID	Params	Subject paths
$P_{3\text{-}0}$	11	$\mathbf{T_0}(P_0)\rightarrow T_4(\text{none})\rightarrow T_1(P_1,P_2)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_3,P_4)\rightarrow T_5(P_5)\rightarrow T_9(P_6)\rightarrow T_7(\text{none})\rightarrow T_{11}(P_7,P_8)\rightarrow T_{12}(P_9,P_{10})$
$P_{3\text{-}1}$	9	$\mathbf{T_1}(P_0,P_1)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_2,P_3)\rightarrow T_5(P_4)\rightarrow T_{10}(P_5)\rightarrow T_{15}(\text{none})\rightarrow T_8(P_6)\rightarrow T_7(\text{none})\rightarrow T_{13}(P_7,P_8)\rightarrow T_{17}(\text{none})$
$P_{3\text{-}2}$	11	$T_0(P_0)\rightarrow \mathbf{T_2}(P_1,P_2)\rightarrow T_{13}(P_3,P_4)\rightarrow T_{14}(P_5,P_6)\rightarrow T_{13}(P_7,P_8)\rightarrow T_{15}(\text{none})\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_0(P_9)\rightarrow T_4(\text{none})\rightarrow T_0(P_{10})$
$P_{3\text{-}3}$	8	$T_1(P_0,P_1)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_0(P_2)\rightarrow \mathbf{T_3}(P_3,P_4)\rightarrow T_{20}(\text{none})\rightarrow T_1(P_5,P_6)\rightarrow T_5(P_7)\rightarrow T_{15}(\text{none})\rightarrow T_7(\text{none})\rightarrow T_{15}(\text{none})$
$P_{3\text{-}4}$	12	$T_0(P_0)\rightarrow \mathbf{T_4}(\text{none})\rightarrow T_1(P_1,P_2)\rightarrow T_5(P_3)\rightarrow T_{10}(P_4)\rightarrow T_7(\text{none})\rightarrow T_7(\text{none})\rightarrow T_{12}(P_5,P_6)\rightarrow T_{14}(P_7,P_8)\rightarrow T_{13}(P_9,P_{10})\rightarrow T_{10}(P_{11})$
$P_{3\text{-}5}$	10	$T_1(P_0,P_1)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_2,P_3)\rightarrow \mathbf{T_5}(P_4)\rightarrow T_{15}(\text{none})\rightarrow T_{10}(P_5)\rightarrow T_8(P_6)\rightarrow T_7(\text{none})\rightarrow T_{11}(P_7,P_8)\rightarrow T_8(P_9)$
$P_{3\text{-}6}$	7	$T_1(P_0,P_1)\rightarrow \mathbf{T_6}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_2,P_3)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_4,P_5)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_0(P_6)\rightarrow T_4(\text{none})$
$P_{3\text{-}7}$	9	$T_0(P_0)\rightarrow T_4(\text{none})\rightarrow T_1(P_1,P_2)\rightarrow T_5(P_3)\rightarrow T_{10}(P_4)\rightarrow T_{11}(P_5,P_6)\rightarrow T_{11}(P_7,P_8)\rightarrow T_{15}(\text{none})\rightarrow T_{15}(\text{none})\rightarrow \mathbf{T_7}(\text{none})\rightarrow T_{16}(\text{none})$
$P_{3\text{-}8}$	9	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_{14}(P_3,P_4)\rightarrow T_{10}(P_5)\rightarrow T_7(\text{none})\rightarrow \mathbf{T_8}(P_6)\rightarrow T_{10}(P_7)\rightarrow T_{15}(\text{none})\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_0(P_8)$
$P_{3\text{-}9}$	11	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{12}(P_3,P_4)\rightarrow T_{14}(P_5,P_6)\rightarrow T_{15}(\text{none})\rightarrow \mathbf{T_9}(P_7)\rightarrow T_{15}(\text{none})\rightarrow T_{10}(P_8)\rightarrow T_{16}(\text{none})\rightarrow T_{20}(\text{none})\rightarrow T_1(P_9,P_{10})$
$P_{3\text{-}10}$	10	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow \mathbf{T_{10}}(P_3)\rightarrow T_{11}(P_4,P_5)\rightarrow T_8(P_6)\rightarrow T_9(P_7)\rightarrow T_7(\text{none})\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_8,P_9)\rightarrow T_6(\text{none})$
$P_{3\text{-}11}$	12	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_7(\text{none})\rightarrow T_9(P_3)\rightarrow T_7(\text{none})\rightarrow T_{10}(P_4)\rightarrow T_8(P_5)\rightarrow T_{13}(P_6,P_7)\rightarrow T_{13}(P_8,P_9)\rightarrow \mathbf{T_{11}}(P_{10},P_{11})\rightarrow T_{16}(\text{none})$
$P_{3\text{-}12}$	10	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow \mathbf{T_{12}}(P_3,P_4)\rightarrow T_9(P_5)\rightarrow T_7(\text{none})\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_0(P_6)\rightarrow T_2(P_7,P_8)\rightarrow T_9(P_9)\rightarrow T_7(\text{none})$
$P_{3\text{-}13}$	10	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow \mathbf{T_{13}}(P_3,P_4)\rightarrow T_{15}(\text{none})\rightarrow T_{13}(P_5,P_6)\rightarrow T_{10}(P_7)\rightarrow T_{13}(P_8,P_9)\rightarrow T_7\ (\text{none})\rightarrow T_7(\text{none})\rightarrow T_{17}(\text{none})\rightarrow T_{18}(\text{none})$
$P_{3\text{-}14}$	12	$T_1(P_0,P_1)\rightarrow T_6(\text{none})\rightarrow T_{18}(\text{none})\rightarrow T_1(P_2,P_3)\rightarrow T_5(P_4)\rightarrow \mathbf{T_{14}}(P_5,P_6)\rightarrow T_{10}(P_7)\rightarrow T_{13}(P_8,P_9)\rightarrow T_7(\text{none})\rightarrow T_{15}(\text{none})\rightarrow T_{13}(P_{10},P_{11})$
$P_{3\text{-}15}$	13	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_{14}(P_3,P_4)\rightarrow T_{15}(\text{none})\rightarrow T_9(P_5)\rightarrow T_7(\text{none})\rightarrow T_{14}(P_6,P_7)\rightarrow T_{13}(P_8,P_9)\rightarrow \mathbf{T_{15}}(\text{none})\rightarrow T_9(P_{10})\rightarrow T_{11}(P_{11},P_{12})$
$P_{3\text{-}16}$	12	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_{12}(P_3,P_4)\rightarrow T_{15}(\text{none})\rightarrow T_{15}(\text{none})\rightarrow T_{11}(P_5,P_6)\rightarrow T_{12}(P_7,P_8)\rightarrow \mathbf{T_{16}}(\text{none})\rightarrow T_{20}(\text{none})\rightarrow T_1(P_9,P_{10})\rightarrow T_5(P_{11})$
$P_{3\text{-}17}$	7	$T_1(P_0,P_1)\rightarrow T_5(P_2)\rightarrow T_{11}(P_3,P_4)\rightarrow T_{10}(P_5)\rightarrow T_{15}(\text{none})\rightarrow T_{15}(\text{none})\rightarrow T_7(\text{none})\rightarrow T_7(\text{none})\rightarrow T_{10}(P_6)\rightarrow \mathbf{T_{17}}(\text{none})\rightarrow T_{18}(\text{none})$
$P_{3\text{-}18}$	10	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{10}(P_3)\rightarrow T_8(P_4)\rightarrow T_{17}(\text{none})\rightarrow \mathbf{T_{18}}(\text{none})\rightarrow T_1(P_5,P_6)\rightarrow T_5(P_7)\rightarrow T_{14}(P_8,P_9)\rightarrow T_7(\text{none})\rightarrow T_{15}(\text{none})$
$P_{3\text{-}19}$	12	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{12}(P_3,P_4)\rightarrow T_7(\text{none})\rightarrow T_{12}(P_5,P_6)\rightarrow T_{13}(P_7,P_8)\rightarrow T_{16}(\text{none})\rightarrow T_{19}(\text{none})\rightarrow T_0(P_9)\rightarrow T_3(P_{10},P_{11})\rightarrow \mathbf{T_{19}}(\text{none})$
$P_{3\text{-}20}$	12	$T_0(P_0)\rightarrow T_2(P_1,P_2)\rightarrow T_{13}(P_3,P_4)\rightarrow T_{14}(P_5,P_6)\rightarrow T_8(P_7)\rightarrow T_7(\text{none})\rightarrow T_{13}(P_8,P_9)\rightarrow T_8(P_{10})\rightarrow T_{16}(\text{none})\rightarrow \mathbf{T_{20}}(\text{none})\rightarrow T_0(P_{11})$

set are 5, 8 and 11 transitions respectively. In this way, we have subject TPs that cover various TPs lengths and also different numbers of required input parameters. The three sets of randomly generated paths through the class 2 transport protocol EFSM are given in Tables 4, 5, and 6 respectively.

As mentioned earlier, In these two EFSMs the maximum level of nested *IF statements* is less or equal to 3 and so we do not apply the conventional approach here since these models will not help contrasting the two performances, nevertheless, they are suitable to evaluate the performance of the proposed approach.

In all tables of subject TPs, each TP is given as a sequence of transition labels together with the parameters required by each transition and the total number of input parameters required

Table 7. The results achieved by the proposed and random techniques on each subject TP from the first set of subjects TPs of INRES protocol.

Path ID	Method	Gen.	Time /m	Input parameters : integer					
				P_0	P_1	P_2	P_3	P_4	P_5
I_{1-0}	Proposed	65	0.04	781	250	1	342	0	363
	Random	1000	0.44	-	-	-	-	-	-
I_{1-1}	Proposed	56	0.04	555	1	138	106	0	381
	Random	1000	0.44	-	-	-	-	-	-
I_{1-2}	Proposed	58	0.04	97	836	1	747	0	-
	Random	1000	0.44	-	-	-	-	-	-
I_{1-3}	Proposed	1000	0.46	-	-	-	-	-	-
	Random	1000	0.44	-	-	-	-	-	-
I_{1-4}	Proposed	46	0.03	253	987	169	1	630	0
	Random	1000	0.44	-	-	-	-	-	-
I_{1-5}	Proposed	74	0.04	922	1	262	0	651	1
	Random	1000	0.44	-	-	-	-	-	-
I_{1-6}	Proposed	57	0.04	869	1	704	2	867	0
	Random	1000	0.46	-	-	-	-	-	-
I_{1-7}	Proposed	37	0.03	246	935	1	667	666	0
	Random	1000	0.44	-	-	-	-	-	-
I_{1-8}	Proposed	1	0.01	25	121	10	15	5	-
	Random	1	0.01	100	10	15	33	47	-
I_{1-9}	Proposed	33	0.03	414	1	220	0	219	-
	Random	1000	0.44	-	-	-	-	-	-
I_{1-10}	Proposed	1	0.01	841	646	140	345	934	-
	Random	1	0.01	999	528	634	397	167	-
I_{1-11}	Proposed	31	0.03	581	1	769	0	958	-
	Random	1000	0.44	-	-	-	-	-	-
I_{1-12}	Proposed	1	0.01	33	15	44	68	-	-
	Random	1	0.01	19	45	67	98	-	-
I_{1-13}	Proposed	31	0.03	921	1	552	726	0	-
	Random	1000	0.44	-	-	-	-	-	-
I_{1-14}	Proposed	35	0.03	678	1	999	0	604	-
	Random	1000	0.45	-	-	-	-	-	-

by each TP. Transitions that do not require input parameters have the label *none* in their parameter section. Once the subject TPs were ready, we applied to each subject TP two search techniques, random and ET-based with the proposed fitness approach.

Both of the ET-based and random techniques were implemented with the publicly available Genetic and Evolutionary Algorithm Toolbox GEATbx [40]. A detailed description of each of the GEATbx parameters used with ET-based technique is beyond the scope of this paper. However, these parameters are fully explained at the GEATbx website [40] and we record the values used here to allow experiments to be replicated. An integer valued encoding was use to represent the input parameters. The population size was 100 individuals where each individual consists of 13 integer variables which represent the maximum number of input parameters required by the considered longest TP. The range of values allowed for each variable was [0-1000]. The selection method was linear-ranking with a selective pressure set to 1.8. Discrete recombination was used to recombine individuals whereas mutate integer method was used for mutation. GEATbx allows the use of standard random approach by setting the recombination and mutation methods to ‘recnone’ and ‘mutrandint’ respectively. The two techniques were given 1000 generations before search was terminated. Also, search termination occurred if the objective value of zero was achieved. Finally, we repeated the search with each technique 10 times for each subject path.

Table 8. The results achieved by the proposed and random techniques on each subject TP from the second set of subjects TPs of INRES protocol.

Path ID	Method	Gen.	Time /m	Input parameters : integer							
				P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
$I_{2\text{-}0}$	Proposed	43	0.06	140	729	1	752	0	-	-	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}1}$	Proposed	37.2	0.04	711	1	765	0	628	134	-	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}2}$	Proposed	101	0.09	800	1	323	0	24	495	1	420
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}3}$	Proposed	59	0.04	830	1	738	0	-	-	-	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}4}$	Proposed	41	0.04	757	1	547	107	0	-	-	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}5}$	Proposed	91	0.08	841	1	177	612	0	648	1	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}6}$	Proposed	46	0.04	181	535	1	430	0	-	-	-
	Random	1000	0.46	-	-	-	-	-	-	-	-
$I_{2\text{-}7}$	Proposed	49	0.04	284	1	947	369	0	439	-	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}8}$	Proposed	8	0.01	924	1	961	184	646	921	658	-
	Random	19	0.03	215	1	785	333	150	633	681	-
$I_{2\text{-}9}$	Proposed	87	0.08	258	1	352	450	0	16	1	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}10}$	Proposed	9	0.01	885	1	322	45	558	-	-	-
	Random	17	0.01	775	1	780	552	350	-	-	-
$I_{2\text{-}11}$	Proposed	81	0.08	859	1	421	55	0	356	1	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}12}$	Proposed	36	0.03	504	1	908	489	819	0	-	-
	Random	1000	0.45	-	-	-	-	-	-	-	-
$I_{2\text{-}13}$	Proposed	35	0.03	706	253	1	709	0	-	-	-
	Random	1000	0.44	-	-	-	-	-	-	-	-
$I_{2\text{-}14}$	Proposed	33	0.03	205	1	400	0	792	356	-	-
	Random	1000	0.44	-	-	-	-	-	-	-	-

4.2. Experiment results

The ET using proposed fitness and random approaches were applied ten times to each subject TP. During the search, we recorded the value of context variables, *State* variable, input parameters and the name of the transition being taken. Also, we recorded the time and the number of generations passed until the search was terminated. Since the complete extended set of results cannot fit in this paper, we only report summaries of the results which show the average number of generations (*Gen.*) in ten tries, average time elapsed, and a set of input parameters values achieved by each applied technique for each subject TP in each set.

4.2.1. Results of INRES Protocol

The results achieved from the first and second set of subject TPs derived from INRES EFSM are reported in Tables 7 and 8. From Table 7, we observe that random search was only successful on 3 TPs where only one generation was required to trigger each of these TPs, however, the proposed technique successfully triggered 14 TPs. There was only one TP ($I_{1\text{-}3}$) that none of the two approaches was successful on. This TP is infeasible due to transition t_3 being called after transition t_0. From Fig.1, t_3 has a guard ($counter \geq 4$), however t_0 has an operation ($counter := 0$) therefore the specified path is infeasible. If we excluded the infeasible path, the proposed approach was successful on all the feasible TPs included in this set.

From Table 8, all the 15 TPs included on the second set were successfully triggered by the

Table 9. The results achieved by the proposed and random techniques on each TP from the first set of subjects TPs of class 2 transport protocol.

Path ID	Method	Gen.	Time/ m	Input parameters : integer								
				P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_{1-0}	Proposed	76.8	0.05	850	626	169	386	0	0	-	-	-
	Random	1000	0.77	-	-	-	-	-	-	-	-	-
P_{1-1}	Proposed	145	0.09	796	618	286	55	83	0	-	-	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-
P_{1-2}	Proposed	264.2	0.18	966	151	548	0	0	0	0	352	-
	Random	1000	0.75	-	-	-	-	-	-	-	-	-
P_{1-3}	Proposed	1	0.01	232	951	21	389	839	871	-	-	-
	Random	1	0.01	117	635	570	248	794	714	-	-	-
P_{1-4}	Proposed	13.7	0.01	111	774	149	958	138	0	-	-	-
	Random	51.9	0.06	586	661	481	750	3	131	-	-	-
P_{1-5}	Proposed	204.9	0.15	844	669	669	83	48	0	0	868	-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-
P_{1-6}	Proposed	1	0.01	638	743	450	985	461	-	-	-	-
	Random	1	0.01	413	423	819	930	763	-	-	-	-
P_{1-7}	Proposed	148.1	0.11	597	176	228	0	7	27	117	-	-
	Random	1000	0.79	-	-	-	-	-	-	-	-	-
P_{1-8}	Proposed	110.1	0.08	700	293	442	1	0	230	-	-	-
	Random	1000	0.79	-	-	-	-	-	-	-	-	-
P_{1-9}	Proposed	69.2	0.05	858	358	678	0	0	446	-	-	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-
P_{1-10}	Proposed	271.6	0.18	829	127	415	0	4	96	0	4	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-
P_{1-11}	Proposed	149.2	0.09	724	78	392	0	1	96	45	-	-
	Random	1000	0.79	-	-	-	-	-	-	-	-	-
P_{1-12}	Proposed	163.9	0.12	895	191	112	93	45	0	0	-	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-
P_{1-13}	Proposed	99.7	0.07	622	744	187	137	5	2	134	-	-
	Random	1000	0.77	-	-	-	-	-	-	-	-	-
P_{1-14}	Proposed	323.4	0.22	613	419	126	0	13	0	0	102	0
	Random	1000	0.70	-	-	-	-	-	-	-	-	-
P_{1-15}	Proposed	18.2	0.01	696	573	10	613	0	-	-	-	-
	Random	84.2	0.04	877	878	844	234	642	-	-	-	-
P_{1-16}	Proposed	145.9	0.12	350	155	590	25	74	0	0	-	-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-
P_{1-17}	Proposed	83	0.05	729	331	354	0	11	-	-	-	-
	Random	1000	0.75	-	-	-	-	-	-	-	-	-
P_{1-18}	Proposed	1	0.01	208	361	826	5	68	-	-	-	-
	Random	1	0.01	622	598	565	410	120	-	-	-	-
P_{1-19}	Proposed	1	0.01	22	779	781	269	416	404	-	-	-
	Random	1	0.01	883	936	443	145	589	842	-	-	-
P_{1-20}	Proposed	1	0.01	500	639	493	321	465	244	-	-	-
	Random	1	0.01	486	736	612	168	174	625	-	-	-

proposed approach, however, random search was only successful on two TPs where it is observed that these two TPs required relatively few generations. Furthermore, the proposed approach performance in terms of the required generations on these two TPs was better than that of random. However, the performance, in terms of the number of successfully triggered TPs, of random search was worse than that observed in the first set which can be related to that fact that the second set has longer TPs than that of the first set.

From both tables, we can state that the majority of TPs were not easy to trigger according to the random search performance. Nevertheless, the proposed technique performed similarly on both sets of TPs and successfully triggered all the feasible ones.

4.2.2. Results of class 2 transport protocol

The results achieved from the three sets of subject TPs derived from the class 2 transport protocol are reported in Tables 9, 10 and 11. From Table 9, we can identify two categories of

Table 10. The results achieved by the proposed and random techniques on each TP from the second set of subjects TPs of class 2 transport protocol.

Path ID	Method	Gen.	Time/ m	Input parameters : integer										
				P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
$P_{2\text{-}0}$	Proposed	365	0.23	869	37	740	81	4	80	24	80	62	0	0
	Random	1000	0.78	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}1}$	Proposed	1	0.01	453	80	509	852	550	126	-	-	-	-	-
	Random	1	0.01	878	886	376	410	596	331	-	-	-	-	-
$P_{2\text{-}2}$	Proposed	95.8	0.07	150	931	765	396	50	153	7	0	9		-
	Random	1000	0.75	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}3}$	Proposed	22.5	0.02	8	827	363	28	785	145	270	1	132	-	-
	Random	52.1	0.03	6	837	118	512	854	115	550	46	84	-	-
$P_{2\text{-}4}$	Proposed	83.2	0.06	521	264	75	342	665	0	0	-	-	-	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}5}$	Proposed	265.5	0.18	955	222	461	0	11	134	0	382	9	122	375
	Random	1000	0.76	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}6}$	Proposed	209.7	0.15	5	160	844	101	332	0	0	1	136		-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}7}$	Proposed	283.7	0.20	1000	292	767	659	0	2	0	810	0	0	-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}8}$	Proposed	307	0.21	581	43	450	608	1	0	877	0	13		-
	Random	1000	0.79	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}9}$	Proposed	235.6	0.16	602	648	116	67	0	0	0	12	840	-	-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}10}$	Proposed	337.3	0.23	970	883	704	108	30	0	12	995	247	2	126
	Random	1000	0.81	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}11}$	Proposed	197.7	0.13	613	173	868	1	5	112	9	-	-	-	-
	Random	1000	0.72	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}12}$	Proposed	211	0.15	456	487	55	0	0	3	140	-	-	-	-
	Random	1000	0.72	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}13}$	Proposed	216.8	0.15	679	248	494	0	0	30	102	379	426	532	-
	Random	1000	0.73	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}14}$	Proposed	87.9	0.05	935	796	625	13	16	42	73	279	883		-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}15}$	Proposed	129	0.08	438	845	435	765	412	85	0	0	0	152	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}16}$	Proposed	1	0.01	885	796	505	19	369	390	395	-	-	-	-
	Random	1	0.01	550	711	265	530	935	100	768	-	-	-	-
$P_{2\text{-}17}$	Proposed	88	0.07	969	49	67	0	1	305	856	-	-	-	-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}18}$	Proposed	145.7	0.09	684	180	697	4	68	0	8	657	966	572	-
	Random	1000	0.76	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}19}$	Proposed	145.2	0.09	507	515	173	621	338	18	90	50	400	0	0
	Random	1000	0.77	-	-	-	-	-	-	-	-	-	-	-
$P_{2\text{-}20}$	Proposed	91.6	0.07	612	81	267	0	290	261	344	551	-	-	-
	Random	109.3	0.07	939	273	544	122	19	305	411	508	-	-	-

subjects TPs, the first contains the TPs that were successfully randomly triggered (7 TPs) whereas the other category includes the rest of the subject TPs (13 TPs) where a random search was unsuccessful. In both categories we observe that the proposed approach was successful and so all the subject TPs included in the first set of subject TPs were triggered successfully. Since not all the paths through an EFSM model have the same level of difficulty, it is natural to have some TPs that are likely to be triggered randomly. This tendency is likely to be observed with short TPs since they have fewer transitions and so fewer guards. For example, if we consider the subject TP P_{1_6} shown in Table 4, we notice that this TP has only one guarded transition t_3 (see Table 1.) and so this TP is likely to be triggered randomly as observed in Table 9. Furthermore, for TPs that were triggered randomly, the proposed approach performance was superior since it always required either the same or fewer generations.

From Table 10, we also observe the same two categories of TPs identified previously: one includes TPs that were triggered randomly (4 TPs) and the other contains the rest of the TPs

Table 11. The results achieved by the proposed and random techniques on each TP from the third set of subjects TPs of class 2 transport protocol.

Path ID	Method	Gen.	Time/m	Input parameters : integer												
				P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
$P_{3\text{-}0}$	Proposed	169.8	0.12	691	863	962	944	4	206	837	0	10	0	231	-	-
	Random	1000	0.75	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}1}$	Proposed	70.7	0.05	884	683	578	689	85	954	0	66	76	-	-	-	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}2}$	Proposed	208.7	0.16	629	102	199	133	0	72	0	4	125	485	300	-	-
	Random	1000	0.76	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}3}$	Proposed	1	0.01	953	545	251	945	835	956	170	386	-	-	-	-	-
	Random	1	0.01	210	469	176	369	652	83	556	67	-	-	-	-	-
$P_{3\text{-}4}$	Proposed	145.4	0.09	551	994	71	551	924	2	392	39	90	142	0	916	-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}5}$	Proposed	237.1	0.17	222	602	1000	438	294	480	0	0	8	1	-	-	-
	Random	1000	0.75	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}6}$	Proposed	1	0.01	797	107	705	66	890	945	484	-	-	-	-	-	-
	Random	1	0.01	108	418	599	24	763	213	286	-	-	-	-	-	-
$P_{3\text{-}7}$	Proposed	220.8	0.16	657	999	488	143	393	0	13	0	0	-	-	-	-
	Random	1000	0.72	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}8}$	Proposed	75.6	0.05	463	674	226	31	66	306	0	565	717	-	-	-	-
	Random	1000	0.78	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}9}$	Proposed	88.4	0.05	1000	537	513	0	438	68	0	223	902	696	879	-	-
	Random	1000	0.72	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}10}$	Proposed	278.2	0.18	869	690	307	1	0	7	0	474	349	273	-	-	-
	Random	1000	0.79	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}11}$	Proposed	355.9	0.23	843	534	261	786	801	0	52	84	132	10	1	776	-
	Random	1000	0.76	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}12}$	Proposed	33.2	0.02	879	573	38	0	60	451	489	163	92	440	-	-	-
	Random	175.4	0.08	692	150	353	0	174	824	546	220	834	909	-	-	-
$P_{3\text{-}13}$	Proposed	191.9	0.08	296	490	0	105	35	2	134	551	26	103	-	-	-
	Random	1000	079	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}14}$	Proposed	204.8	0.15	226	766	830	45	148	5	45	928	79	61	69	482	-
	Random	1000	0.76	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}15}$	Proposed	290.2	0.18	786	347	353	38	0	123	29	1	4	133	1	764	312
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}16}$	Proposed	321	0.22	886	250	270	0	253	0	1	0	1000	832	470	298	-
	Random	1000	0.74	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}17}$	Proposed	105	0.06	959	508	940	0	5	887	491	-	-	-	-	-	-
	Random	1000	0.77	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}18}$	Proposed	90	0.06	876	368	739	103	0	313	598	54	116	0	-	-	-
	Random	1000	0.75	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}19}$	Proposed	211.6	0.15	999	704	74	0	308	0	561	143	0	89	112	289	-
	Random	1000	0.76	-	-	-	-	-	-	-	-	-	-	-	-	-
$P_{3\text{-}20}$	Proposed	1000	0.80	-	-	-	-	-	-	-	-	-	-	-	-	-
	Random	1000	0.77	-	-	-	-	-	-	-	-	-	-	-	-	-

where a random search was unsuccessful (16 TPs). In both categories, the proposed approach was successful and so all the TPs included in the second set were triggered. Since the second set of subject TPs included longer TPs than the first set, it was not surprising that fewer TPs were randomly triggered. However, the performance of the proposed approach was the same since it triggered all of the subject TPs. Furthermore, the proposed approach performance on the subject TPs that were randomly triggered was also superior since it required always same or fewer generations than a random search.

From Table 11, we can divide the results into three categories: the first category contains TPs that were randomly triggered (3 TPs), the second category contains TPs where a random search was unsuccessful (16 TPs) and the proposed approach was successful whereas the third category includes TPs where the two applied techniques were unsuccessful (only the last TP P_{3_20}). In the first two categories, we observe that the proposed approach produced a valid input test data that successfully triggered the subject TPs (19 TPs). Furthermore, since TPs included in the third set

Table 12. The success rate of the two applied techniques on the subject EFSM models

Subject EFSM	Subject TPs	Num of TPs	Method	Success (TPs)	Fail (TPs)	Success rate %
INRES protocol	First set	14*	Proposed	14	0	100 %
			Random	3	11	21.4 %
	Second set	15	Proposed	15	0	100 %
			Random	2	13	13.3 %
class 2 transport protocol	First set	21	Proposed	21	0	100 %
			Random	7	14	33.3 %
	Second set	21	Proposed	21	0	100 %
			Random	4	17	19 %
	Third set	20*	Proposed	20	0	100 %
			Random	3	17	15 %
Both protocols	All sets	91*	Proposed	91	0	100 %
			Random	19	72	20.8 %

* Infeasible paths are excluded

consists of more transitions than these included in the first and second sets, it was not surprising to observe that a random search performance was worse that that exhibited previously (only 3 TPs were triggered randomly). Nevertheless, the proposed approach exhibited similar performance to that observed from previous results and this enhances the confidence about the validity of the proposed fitness calculation.

For the third category of TPs reported in Table 11 where both techniques failed, there was only one such TP P_{3_20}. This path is infeasible due to the transition t_8 being called before transition t_{10}. From Table 1, the transition t_8 has the condition ($R\_credit \neq 0$), where t_{10} is the only transition that assigns a value, different from zero, to the context variable *R_credit*. Thus, if the occurrence of transition t_8 is earlier than the occurrence of transition t_{10}, the resultant path is infeasible.

From Tables 7, 8, 9, 10, and 11, it is clear that the majority of the subject TPs were not easy to trigger. From Table 7, only 3 TPs were triggered by random search and this gives a success rate of 21.4 % with the infeasible path being excluded. However, this rate dropped to 13.3 % as observed in Table 8. Similarly from Table 9, there are 7 TPs that triggered successfully with random approach and so the success rate is 35%. The success rate of random approach dropped to 20% as observed in Table 10 and to 15.7 % in Table 11. If we exclude the infeasible TPs reported in Tables 7 and 11 all the 91 feasible subject TPs were successfully triggered by the proposed approach with a success rate 100 %, however, only 19 TPs were triggered by random approach with a success rate of 20.8 % as shown in Table 12.

Finally, the cost introduced by the proposed approach in terms of time required to perform was relatively small. In the worst case, the time required by the proposed approach did not exceed 1 minute, however, if we consider the experiment environment (*Matlab* and *GEATbx*) it is very likely that the observed cost can be further reduced with a stand alone evolutionary testing tool.

4.3. Threats to validity

In this subsection we discuss the potential threats to the validity of our study.

Threats to internal validity are the factors that can affect our results without our interference or our knowledge. To reflect this on our study, this can be related to the way that the subject paths have been constructed. That is, since a path is randomly generated, we do not control the way in which transitions have appeared in the path. This results in the guards associated with each transition have been sequenced in a particular order. To limit the problems related to this, we generated five sets of subject paths derived from two EFSM case studies in order to cover a wider range of paths. Furthermore, we applied our technique ten times for each subject path to

make it less likely that what we observed is an extraordinary phenomenon.

Threats to external validity are the conditions that restrict our ability to generalize our results. The significant threats to external validity of this study are related to: First, the subject EFSM models, though nontrivial, they have a relatively small number of states and transitions, and larger EFSM model might be helpful to derive more general conclusion about the validity of the proposed approach. Nevertheless, this threat has been limited by using the same EFSM models used to evaluate other EFSM testing techniques. Second, the lengths of subject paths covered in this study are selected to cover three different path lengths. Longer path length may be subject to a different cost e.g., more generations are needed in order to traverse a path. Additional experiments could investigate the impact of using longer paths.

5. Conclusion

Due to the complexity of EFSM model, automating the process of input test data generation for a set of feasible paths is a challenging task. In order to address this problem, we present an effective and easy to implement approach based on ET technique that enables the automatic generation of input test data to trigger a given feasible path through an EFSM model.

The approach treats transitions in an EFSM model as a set of functions and the problem of path test data generation is a search for a suitable set of input parameters to be applied to a set of functions that are called sequentially. Since the conventional ET approach is mainly effective in structural testing where a test target is represented as a single node, a new fitness calculation method was introduced to accommodate the problem of path test data generation where there is a set of subtargets (each transition in a path) to be achieved. The new fitness method can be seen as applying the method of Wegener et al. [32] two times. The first time is to each transition in order to achieve a subtarget and the second time is to the whole path where each guarded transition is considered as a conditional node. We successfully utilized our approach to generate transition coverage test suites for the INRES and class 2 transport protocols.

In our experiment, we randomly generated five sets of paths with different lengths through the considered EFSM models and then we applied the proposed approach together with the random technique. For all the feasible paths, the proposed approach was superior and located test data that trigger the path and so had a success rate of 100%. However, the random search failed in most of the cases and had a success rate of 20.8%. The empirical results provided strong evidence that the proposed approach provides a fitness feedback which can progress ET search successfully.

The method presented in this paper is complementary to other available EFSM testing techniques that function by generating a set of paths through a given EFSM model since it can be incorporated in the input test data generation phase.

Further research will focus on employing evolutionary testing to estimate the feasibility of transition paths in order to automate the process of feasible transition paths generation. This includes constructing a fitness metric that depends on analysing the dependencies among transitions' actions and conditions to estimate the likelihood that a given transition path is feasible. Such an approach together with the one proposed in this paper can form an integrated method to automate testing from EFSM models.

References

[1]. Korel, B., *Automated software test data generation.* Software Engineering, IEEE Transactions on, 1990. **16**(8): p. 870-879.

[2]. McMinn, P., *Search-based software test data generation: a survey: Research Articles.* Software Testing, Verification & Reliability, 2004. **14**(2): p. 105-156.

[3]. Harman, M. *Automated Test Data Generation using Search Based Software Engineering*. in *Automation of Software Test , 2007. AST '07. Second International Workshop on*. 2007.

[4]. Michael, C.C., G. McGraw, and M.A. Schatz, *Generating software test data by evolution.* Software Engineering, IEEE Transactions on, 2001. **27**(12): p. 1085-1110.

[5]. Petrenko, A., S. Boroday, and R. Groz, *Confirming configurations in EFSM testing.* Software Engineering, IEEE Transactions on, 2004. **30**(1): p. 29-42.

[6]. Hierons, R.M., *Separating sequence overlap for automated test sequence generation.* Automated Software Engineering., 2006. **13**(2): p. 283-301.

[7]. Derderian, K., R.M. Hierons, M. Harman, and Q. Guo, *Automated Unique Input Output Sequence Generation for Conformance Testing of FSMs.* The Computer Journal, 2006. **49**(3): p. 331-344.

[8]. Lai, R., *A survey of communication protocol testing.* Journal of Systems and Software, 2002. **62**(1): p. 21-46.

[9]. Lee, D. and M. Yannakakis, *Principles and methods of testing finite state machines-a survey.* Proceedings of the IEEE, 1996. **84**(8): p. 1090-1123.

[10]. Duale, A.Y. and M.U. Uyar, *A method enabling feasible conformance test sequence generation for EFSM models.* Computers, IEEE Transactions on, 2004. **53**(5): p. 614-627.

[11]. Duale, A.Y., M.U. Uyar, B.D. McClure, and S. Chamberlain. *Conformance testing: towards refining VHDL specifications*. in *Military Communications Conference Proceedings, 1999. MILCOM 1999. IEEE*. 1999.

[12]. Chanson, S.T. and J. Zhu. *A unified approach to protocol test sequence generation*. in *INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. IEEE*. 1993.

[13]. Hierons, R.M., T.-H. Kim, and H. Ural, *On the testability of SDL specifications.* Computer Networks, 2004. **44**(5): p. 681-700.

[14]. Tahat, L.H., B. Vaysburg, B. Korel, and A.J. Bader. *Requirement-based automated black-box test generation*. in *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*. 2001.

[15]. Petrenko, A., G.v. Bochmann, and M. Yao, *On fault coverage of tests for finite state specifications.* Computer Networks and ISDN Systems, 1996. **29**(1): p. 81-106.

[16]. Ramalingom, T., K. Thulasiraman, and A. Das, *Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines.* Computer Communications, 2003. **26**(14): p. 1622-1633.

[17]. Derderian, K., R.M. Hierons, M. Harman, and Q. Guo, *Generating feasible input sequences for extended finite state machines (EFSMs) using genetic algorithms,* in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. 2005, ACM: Washington DC, USA.

[18]. Ramalingom, T., K. Thulasiraman, and A. Das. *Context independent unique sequences generation for protocol testing*. in *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*. 1996.

[19]. Koh, L.-S. and M.T. Liu. *Test path selection based on effective domains*. in *Network Protocols, 1994. Proceedings., 1994 International Conference on*. 1994. Boston, MA.

[20]. Sarikaya, B., G.v. Bochmann, and E. Cerny, *A Test Design Methodology for Protocol Testing.* Software Engineering, IEEE Transactions on, 1987. **SE-13**(5): p. 518-531.

[21]. Wang, C.-J. and M.T. Liu. *Generating test cases for EFSM with given fault models*. in *INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. IEEE*. 1993.

[22]. Ural, H. and B. Yang, *A test sequence selection method for protocol testing.* Communications, IEEE Transactions on, 1991. **39**(4): p. 514-523.

[23]. Hogrefe, D., *OSI formal specification case study: the Inres protocol and service. Technical Report IAM-91-012*. 1991, University of Bern, Institute of Computer Science and Applied Methematics. p. 5.

[24]. Bochmann, G.V., *Specifications of a simplified transport protocol using different formal description techniques.* Computer Networks and ISDN Systems, 1990. **18**(5): p. 335-377.

[25]. Shih, C.-H., J.-D. Huang, and J.-Y. Jou. *Stimulus generation for interface protocol verification using the nondeterministic extended finite state machine model*. in *High-Level Design Validation and Test Workshop, 2005. Tenth IEEE International*. 2005.

[26]. Holland, J.H., *Adaptation in natural and artificial systems*. 1992, Cambridge, MA: MIT Press. 211.

[27]. Liepins, G.E. and M.R. Hilliard, *Genetic algorithms: Foundations and applications.* Annals of Operations Research, 1989. **21**(1): p. 31-57.

[28]. Yao, X. *Global optimisation by evolutionary algorithms*. in *Parallel Algorithms/Architecture Synthesis, 1997. Proceedings. Second Aizu International Symposium*. 1997.

[29]. Srinivas, M. and L.M. Patnaik, *Genetic algorithms: a survey.* Computer, 1994. **27**(6): p. 17-26.

[30]. Baresel, A., D. Binkley, M. Harman, and B. Korel, *Evolutionary testing in the presence of loop-assigned flags: a testability transformation approach*, in *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*. 2004, ACM: Boston, Massachusetts, USA.

[31]. Tracey, N., J. Clark, K. Mander, and J. McDermid. *An automated framework for structural test-data generation*. in *Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on*. 1998.

[32]. Wegener, J., A. Baresel, and H. Sthamer, *Evolutionary test environment for automatic structural testing.* Information and Software Technology, 2001. **43**(14): p. 841-854.

[33]. Cheng, K.-T. and A.S. Krishnakumar, *Automatic generation of functional vectors using the extended finite state machine model.* ACM Transactions on Design Automation of Electronic Systems., 1996. **1**(1): p. 57-79.

[34]. Dahbura, T.A., K.K. Sabnani, and M.U. Uyar, *Formal methods for generating protocol conformance test sequences.* Proceedings of the IEEE, 1990. **78**(8): p. 1317-1326.

[35]. Budkowski, S. and P. Dembinski, *An introduction to Estelle: a specification language for distributed systems.* Computer Networks and ISDN Systems., 1987. **14**(1): p. 3-23.

[36]. Hierons, R.M. and M. Harman, *Testing conformance of a deterministic implementation against a non-deterministic stream X-machine.* Theoretical Computer Science, 2004. **323**(1-3): p. 191-233.

[37]. Lefticaru, R. and F. Ipate, *Automatic State-Based Test Generation Using Genetic Algorithms*, in *Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. 2007, IEEE Computer Society.

[38]. Lefticaru, R. and F. Ipate, *Functional Search-based Testing from State Machines*, in *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*. 2008, IEEE Computer Society.

[39]. Liu, X., M. Zhang, Z. Bai, L. Wang, W. Du, and Y. Wang. *Function Call Flow based Fitness Function Design in Evolutionary Testing*. in *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*. 2007.

[40]. Pohlheim, H. *GEATbx - Genetic and Evolutionary Algorithm Toolbox for Matlab*. 1994-2008 [cited; Available from: http://www.geatbx.com.