Soft Computing manuscript No.
(will be inserted by the editor)

Using genetic algorithms to generate test sequences for complex timed systems

Alberto Núñez · Mercedes G. Merayo · Robert M. Hierons · Manuel Núñez

Received: date / Accepted: date

Abstract The generation of test data for state based specifications is a computationally expensive process. This problem is magnified if we consider that time constraints have to be taken into account to govern the transitions of the studied system. The main goal of this paper is to introduce a complete methodology, supported by tools, that addresses this issue by representing the test data generation problem as an optimisation problem. We use heuristics to generate test cases. In order to assess the suitability of our approach we consider two different case studies: a communication protocol and the scientific application BIPS3D. We give details concerning how the test case generation problem can be presented as a search problem and automated. Genetic algorithms (GAs) and random search are used to generate test data and evaluate the approach. GAs outperform random search and seem to scale well as the problem size increases. It is worth to mention that we use a very simple fitness function that can be easily adapted to be used with other evolutionary search techniques.

Keywords formal testing · genetic algorithms · timed systems

This research was partially supported by the Spanish MEC project TESIS (TIN2009-14312-C02-01) and the UK EPSRC project Testing of Probabilistic and Stochastic Systems (EP/G032572/1).

A. Núñez, M. G. Merayo, M. Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain,
alberto.nunez@pdi.ucm.es, mgmerayo@fdi.ucm.es, mn@sip.ucm.es
· R. M. Hierons
Department of Information Systems and Computing,
Brunel University
Uxbridge, Middlesex, UB8 3PH United Kingdom,
rob.hierons@brunel.ac.uk

1 Introduction

In order to improve the reliability of complex systems, it is important to use sound methods to formally specify their critical components. The use of formal specifications allows the developers to better understand and model the behaviour of the produced systems. Communication protocols and control systems, amongst others, have used formal specifications based on the classical Finite State Machines (FSMs) formalism. In particular, FSMs have been used to formally model systems in different areas like sequential circuits [17], software development [5] and communication protocols [4]. Unfortunately, it cannot be guaranteed that system implementations fully comply to the specifications and, therefore, it is necessary to fix a *validation* process to ensure that the (behavior of the) implementation conforms to the specification. Among the methods used to check that system implementations behave as expected, *testing* [43,2] is the most widely used in industrial environments. Currently, testing is an important part of the system development process that aims to increase the reliability of the implementation. However, testing can be very expensive and, due to its mainly manual application, prone to errors. This motivates the research in the combination of formal methods and testing [19,18] since progress in this line of work helps to (partially) automatize the testing process with the use of suitable tools [54]. In fact, a very recent study clearly shows that the use of formal methods reduces the cost of complex projects in both time and monetary terms [14]. Other studies advocate for the use of formal approaches to test and validate critical systems in domains such as aeronautics and automotive [27,3].

Testing systems formally specified using FSMs and the automation of the test data generation have been of

interest [29,48,12]. In software testing one of the aims is to distinguish between the correct and incorrect behaviour of system implementations. While the analysis of some systems only considers the correctness of their results (outputs), other systems might have specific time requirements as well. This can be especially relevant in environments where system resources are shared. In fact, in real-time systems the time it takes for a process to execute can be as important as the output. Thus, formal testing techniques started to deal also with this kind of aspects. In fact, there are already a myriad of proposals for formal testing of timed systems and we have been very active in this line of work [36,35, 20,33,37]. Previous approaches to formally test timed systems concentrated on general frameworks. However, they usually did not deal with the problem of generating test sequences that can either focus on an important property or on a specific part of the system under test. The problem of generating test sequences for non-timed systems is not trivial, due to state explosion, and the inclusion of time complicates the testing problem as we will discuss later. Therefore, it is necessary to work with heuristic techniques that can find *good enough* test sequences. There has been some work on using artificial intelligence techniques, like genetic algorithms (GAs) and other meta-heuristic algorithms, to automate software testing [23,38,32,16,47,31]. Our group has lately been very active in this area of research [24,40,9,25,41, 42]. In this paper we concentrate on the use of GAs to generate test sequences in complex timed systems. In short, a GA is a heuristic optimisation technique which derives its behaviour from a metaphor of the processes of evolution in nature [13,51]. GAs have been widely used in search optimisation problems. GAs are known to be particularly useful when searching large, multimodal and unknown search spaces since one of its benefits is their ability to escape local minima in the search for the global minimum.

The initial step of our methodology considers the generation of test sequences for Extended Finite State Machines (EFSMs) models [8,9], where GAs are used to deal with the inherently exponential nature of exploring the combination of constraints associated with transitions of a path in a model. This problem is even more complicated in the case of timed systems. One of the main problems to overcome is that it is not enough to test whether the implementation system is doing what it is supposed to do. Additionally, it is necessary to test that it is also taking the specified time to complete the considered task. Moreover, the tests applied to system implementations have to consider *when* to test [34]. Specifically, this paper addresses the issues related to generating test sequences for temporally constrained EFSM based systems. It focuses on generating *transition paths* with specific properties that can, in turn, be used to generate test input. The problem of generating these paths is represented as a search problem, being GAs used to help to automate the test data generation process. Moreover, simulation techniques have been jointly used with GAs to model and generate test sequences for a complex scientific application.

In order to validate the usefulness of our methodology we considered two case studies of very different nature. The first case study is a basic communication protocol: the class 2 transport protocol. This case study allowed us to validate the main features of our methodology in a simple and easy to control environment. However, it was necessary to consider a more complex system to evaluate whether our methodology scales well. Therefore, we considered a real application: BIPS3D. Briefly, BIPS3D [30] is a scientific application that performs 3-dimensional simulations of BJT and HBT bipolar devices. This case study shows that our methodology indeed scales and it is interesting for itself because it combines, in an intricate way, three different fields of study: *Formal Testing*, by using methods to generate test sequences from formal models, *Artificial Intelligence*, by using GAs to choose the most promising paths to be explored, and *Simulation*, by considering a complex simulator to help in the computation of relevant quantities.

This paper represents an extended and enhanced version of previous work on the generation of test sequences for timed systems [10,11]. We have revised the theoretical framework to fix some minor mistakes. We have included a long discussion, Section 5, about High Performance Computing and the use of simulation techniques as a way to implement our approach. The first case study, the communication protocol, was already introduced in the previously mentioned work, but the complex case study presented in this paper is completely new. Similarly, the integration of the different GAs considered in this paper in the SIMCAN simulation platform [44] is also new and, indeed, represents a novel and interesting combination of formal testing, GAs and simulation.

The rest of the paper is structured as follows. Section 2 describes the problem faced in this paper. Section 3 shows the model of Timed extended finite state machine (TEFSM). Section 4 describes the use of genetic algorithms to aid test case generation. Section 5 briefly enumerates the main characteristics of High Performance Computing (HPC), used in our second case study, and the use of simulation techniques in HPC. Section 6 presents two different study cases to validate

this work. Finally, Section 7 presents our conclusions and some directions for future work.

2 Description of the problem

In this section we introduce the problem that we want to confront and motivate why this problem is relevant. Formal definitions of all the concepts informally used in this section will be given in the next section.

As we have already mentioned in the introduction of the paper, FSMs are known to model appropriately sequential circuits and control portions of communication protocols. However ordinary FSMs are not powerful enough for some applications where extended finite state machines (EFSMs) are used instead. Finite state systems are usually modelled using Mealy machines that produce an output for every transition triggered by an input. In EFSMs however the transitions are associated with conditions. EFSMs have been widely used in the telecommunications field, and are also now being applied to a diverse number of other areas ranging over aircraft, train control, medical and packaging systems. Examples of languages based on EFSMs include SDL, Estelle and Statecharts. In our framework we consider EFSMs with the addition of *time*: timed EFSMs (TFSMs).

Usually the implementation of a system specified by an FSM or EFSM is tested for conformance by applying a sequence of inputs and verifying that the corresponding sequence of outputs is that which is expected. This commonly involves executing a number of transition paths, until all transitions have been tested at least once. In EFSMs, in order to execute a transition path it is necessary to satisfy all of the transition guards involved, in addition to using a specific input sequence to trigger these transitions.

When FSM based systems are tested for conformance with their specification, often a fault can be categorised as either an output fault (wrong output is produced by a transition) or a state transfer fault (the state after a transition is wrong). A test strategy would involve moving M to a state s_i, applying some input x, verifying that the output is y as expected for that transition under test, and using a state verification technique to verify the transition's end state. To achieve this there are test sequences with different properties. State verification sequences for example aim to check if the end state of a transition (or TP) is the one expected and state identification sequences identify the unknown state the system is in [29].

When TEFSMs are tested for conformance there are time related faults that could be present besides the output and transition faults. The time related faults arise when a transition within the implementation takes longer to complete than the time specified by the TEFSM.

In EFSMs (and TEFSMs) test sequence generation is more complex than it is for FSMs. In FSMs all paths are feasible since there are no guards and actions do not affect the traversal [12]. With EFSMs (and TEFSMs), however, in order to trigger the transition path it is necessary to satisfy the transition guards. A transition guard may refer to the values of the internal variables and the input parameters, which in turn can assume different values after each transition. Some transition paths might have no conditions, some might have conditions that are rarely satisfied and some transition paths will be infeasible. The existence of infeasible transition paths creates difficulties for automating the test generation process for EFSMs (and TEFSMs).

With TEFSMs however there is the element of time that needs to be considered as well, making the triggering of a path successfully harder than in EFSMs. One way of approaching the test sequence generation problem is to abstract away the data part of the TEFSM and consider it as an FSM on its own. However, a transition sequence for the underlying FSM of a TEFSM is not guaranteed to be feasible for the actual TEFSM nor to satisfy the temporal constraints. This leads to the problem that an input sequence will only trigger a specific TP in a TEFSM if all the transition guards allow this and so an input sequence generated on the basis of the FSM may not follow the required path in the TEFSM. Another approach is to expand a TEFSM to an FSM and then use the techniques used for FSMs. However this can lead to a combinatorial explosion.

The problem of TP feasibility in EFSMs has been considered in the past [8,9]. That work uses the TP feasibility approach proposed in that work and extends it to consider TEFSMs and problems associated to testing the compliance of an implementation to its temporal aspects of the specification. In addition to estimating the feasibility of a TP in this paper we examine how to consider the temporal properties of a TP and help in related test case generation.

In order to generate a test case for a TEFSM M we can first consider the properties of the TP that this test case is expected to trigger. The general problem of finding a (an arbitrary) feasible transition sequence for a TEFSM is uncomputable, as is generating the necessary input sequence to trigger such a transition sequence. The task of finding a transition sequence with particular temporal conditions complicates the problem even further. Test generation for EFSMs [28,12] and TEFSMs [34] is still an open research problem.

While a random algorithm could be used it does not always produce acceptable results. Test case generation and optimisation for FSM based systems has been of interest [21,12,49,22]. Heuristic search techniques like genetic algorithms (GAs) have been used in problems like estimating feasibility of TPs in EFSMs [9], generating test sequences for FSMs [15,8] and communicating FSMs [7]. Heuristic search techniques can also be applied to the problem of generating TPs with other characteristics if a robust fitness function can be defined. Hence a function that can estimate the likelihood that a TP in a TEFSM can be executed and also satisfies the temporal constraints imposed may help in areas like test sequence generation. Heuristic search can be used to direct a search towards TPs that are likely to satisfy the set requirements.

3 Timed Extended Finite State Machines: model and test sequences

In this section we formally introduce the concepts that will be used along the paper. In particular, we will introduce the notions of timed extended finite state machine and timed feasible transition path.

3.1 Definition of the TEFSM model

EFSMs are Mealy (finite state) machines with parameterised input and output, internal variables, operations and predicates defined over internal variables and input parameters. Timed EFSMs (TEFSMs) are classical EFSMs with added conditions on the time transitions need to take to complete.

We assume that the number of different variables is m. If we assume that each variable x_i belongs to a domain D_i thus the values of all variables at a given point of time can be represented by a tuple belonging to the cartesian product of $D_1 \times D_2 \times ... \times D_m = \Delta$. Regarding the domain to represent time we define that time values belong to a certain domain `Time`.

Definition 1 A TEFSM M can be defined as a tuple $(S, s_0, V, \sigma_0, P, I, O, T, C)$ 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, σ_0 denotes the mapping from the variables in V to their initial values, P is the set of input and output parameters, I is the set of input declarations, O is the set of output declarations, T is the finite set of transitions and C is such that $C \in \Delta$.

A transition $t \in T$ is defined by $(s_s, g_I, g_D, g_C, op, s_f)$ where s_s is the start state of t; g_I is the input guard expressed as (i, P^i, g_{P^i}) where $i \in I \cup \{NIL\}$; $P^i \subseteq P$; and g_{P^i} is the input parameter guard that can either be NIL or be a logical expression in terms of variables in V' and P' where $V' \subseteq V$, $\emptyset \neq P' \subseteq P^i$; g_D is the domain guard and can be either NIL or represented as a logical expression in terms of variables in V' where $V' \subseteq V$; $g_C : \Delta \rightarrow$ `Time` is the time the transition needs to take to complete; *op* is the sequential operation which is made of simple output and assignment statements; and s_f is the final state of t.

The label of a transition in a TEFSM has two guards that decide the feasibility of the transition: the input guard g_I and the domain guard g_D. In order for a transition to be executed g_I, the guard for inputs from the environment must be satisfied. Some inputs may carry values or specific input parameters and M may guard those values with the input parameter guard g_P. Hence the values of the input parameters may determine whether a transition is executed and affect the output of M. The input guard $(NIL, \emptyset, NIL)$ represents no input being required (spontaneous transition). g_D is the guard, or precondition, on the values of the system variables (e.g. $v > 4$, where $v \in V$). Note that in order to satisfy the domain guard g_D of a transition t, it might be necessary to have taken some specific path to the start state of t. *op* is a set of sequential statements such as $v := v + 1$ and $!o$ where $v \in V$, $o \in O$ and $!o$ means 'output o to the environment'. Literal outputs (output directly observable by the user) are denoted with ! and output functions (an output function may produce different output depending on the parameters it receives) without it (e.g. $!o$ and $u(v)$). The operation of a transition in a TEFSM has only simple statements such as output statements and assignment statements, no branching statements are allowed.

In a TEFSM the time a transition took to complete is also important. The time a transition needs to be completed by, represented by g_C can be dependant on the current values of V' and P^i, where $V' \in V$ and $P^i \subseteq P$.

We assume that none of the spontaneous transitions in a TEFSM are without any guards, $g_I = (NIL, \emptyset, NIL)$ and $g_D = NIL$, because they will be uncontrollable. When a transition in a TEFSM is executed, all the actions of the operation specified in its label are performed consecutively and only once and the transition must not take more than g_C time units to perform. The transition is considered to have failed (in terms of compliance to the specification or be considered void) if it is not completed within the required time. Such transitions may be considered void if they take longer than expected in some systems and in others they might still be executed even though they took longer than expected. Our TEFSM model and our work is applicable

to both, however in our case study we considered the former.

Definition 2 A TEFSM M is deterministic if any pair of transitions t and t' initiating from the same state s that share the same input x have mutually exclusive guards.

Upon an input, a TEFSM triggers a given transition depending not only on the input, but on the values of the internal variables. Hence TEFSM may be called dynamically deterministic. As stated before triggering a transition in a TEFSM involves satisfying all those guards and conditions. This makes triggering a transition in TEFSMs more complex than in FSMs and EFSMs. As an example consider the Class 2 transport protocol [50] represented as a TEFSM in Figure 1 and the transition table in Table 1. There are two transitions initiating from state S_2 with the input declaration $N?TrCC$: the transitions t_2 and t_3. However they have mutually exclusive conditions: $opt\_ind \leq opt$ and $opt\_ind > opt$, respectively.

Definition 3 A TEFSM is strongly connected if for every ordered pair of states (s, s') there is some feasible path from s to s'.

We assume that any TEFSM considered is deterministic and strongly connected. For example, consider the Class 2 transport protocol [50] represented as a TEFSM in Figure 1. There is a TP from the initial state S_1 to every other state. TEFSMs (inherently from EFSMs) have a notion knows as configuration that defines the subset of available transitions that can be triggered at any point.

Definition 4 A configuration for a TEFSM M is a combination of state and values of the internal variables V of M.

A TEFSM starts in its initial configuration and there is a configuration for every combination of state s and values of the set internal variables V.

3.2 Transition paths in TEFSMs

Now consider the problem of finding an input sequence that triggers a timed feasible transition path (TFTP) from state s_i to state s_j of a TEFSM M.

Definition 5 A timed feasible transition path (TFTP) for state s_i to state s_j of a TEFSM M is a sequence of transitions initiating from s_i that is feasible for at least one combination of values of the finite set of internal variables V (configuration) of M and ends in s_j.

State identification and state verification sequences for a TEFSM must trigger a TFTP (where the end state is not important). Methods that can help identify TFTPs can potentially be used to help in the state identification and state verification sequence generation problems.

In a transition path for FSMs each transition can be identified and thus represented by its start state and input (s_s, i). However with TEFSMs (as well as EFSMs) this information is not sufficient because there can be more than one transitions sharing the same start state and input due to having mutually exclusive guards. Instead a transition t in a TEFSM M can be identified from its start state, input declaration, input parameter, the input parameter guard and the domain guard $(s_s, i, P^i, g_{P^i}, g_D)$. g_{P^i} and g_D for a transition t in M can both be logical expressions and their results may depend on input parameter P^i of t and the values of some of the internal variables of M. A transition in a transition path of a TEFSM can be identified by a tuple (s_s, i, g_{P^i}, g_D) in which s_i is its start state, i is its input, g_{P^i} is its input guard and g_D is its domain guard. The input parameter P^i is not required in order to be able to uniquely identify a transition in M. Note how in this case some transitions with different domain guards share a common input predicate guard.

Transitions sharing the same start state and input declaration can be classified according to their input guard predicate and domain guard predicate. These predicates consist of logical expressions that can evaluate to either $True$ or $False$ depending on the input parameters or internal variables of M. To identify these for every set of transitions sharing the same start state s and input declaration i, the number of unique input guards (input predicate branches) and unique domain guards (domain predicate branches) is counted and a predicate dependency tree for state s and input declaration i can be constructed.

Consider the Sending state (S_4) in the EFSM M_1 on Figure 1 . There are two transitions initiating from this state that share the same input declaration $N?TrAK$ and input parameters $XpSsq$ and cr. These transitions have the same input but differ only in their input parameter guards and domain guards. Hence a transition in a transition path of a TEFSM can be identified by a tuple (s_s, i, g_{P^i}, g_D) in which s_i is its start state, i is its input, g_{P^i} is its input guard and g_D is its domain guard. The input parameter P^i is not required in order to be able to uniquely identify a transition in M. Note how in this case some transitions with different domain guards share a common input predicate guard.

Not all transitions in TEFSMs have input parameter guards and domain guards and so transitions in

a TEFSM M can be categorised in the following way: *simple transitions* are those transitions that have *no* input parameter guard and *no* domain guard, $g_{P^i} = NIL$ and $g_D = NIL$; g_{P^i} *transitions* are those transitions that *have* input parameter guard but *not* a domain guard, $g_{P^i} \neq NIL$ and $g_D = NIL$; g_D *transitions* are those transitions that *have* a domain guard but *not* an input parameter guard, $g_D \neq NIL$ and $g_{P^i} = NIL$; g_{P^i}-g_D *transitions* are those transitions that *have* both an input parameter guard and a domain guard, $g_{P^i} \neq NIL$ and $g_D \neq NIL$.

An assignment statement in the *op* part of a transition would not generate any observable output. However such assignment statements can still contribute to the value of the output generated in a later transition if the assignment changes the value of one of the output function's parameters. It can also affect the feasibility of the remaining transitions in the transition path and should also be considered. In this paper we consider the importance that the *op* part of the transition can have on the feasibility in TPs but leave this for future work.

Definition 6 An input sequence (IS) is a sequence of input declarations $i \in I$ with associated input parameters $P^i \subseteq P$ of a TEFSM M.

Instead of using g_{P^i} and g_D notations together in order to identify a transition we can simply use a label.

Definition 7 A predicate branch (PB) is a label that represents a pair of g_{P^i} and g_D for a given state s and input declaration i. A PB identifies a transition within a set of transitions with the same start state and input declaration.

PBs can label conditional transitions and be used to help simulate the behaviour of a potential input sequence for a TEFSM without the feasibility restrictions.

Definition 8 An abstract input sequence (AIS) for M represents an input declaration sequence with associated PBs that triggers a TP in the abstracted M.

The advantages of using AIS and simulating the execution of a TEFSM is that the configuration of the TEFSM is not considered. Hence transition traversal evaluations that can be used to estimate the characteristics of a TP can be done without complex computation.

4 Using genetic algorithms to aid test case generation

In this section we present our approach to use GAs for generating test data and evaluate the approach. First, a brief overview about genetic algorithms is presented. The two GAs used in this work are also described. Second, we describe the definition of the fitness function used in this work.

4.1 Overview of Genetic Algorithms

A *Genetic Algorithm* [13,51] is a heuristic optimisation technique which derives its behaviour from a metaphor of the processes of evolution in nature. As we indicated in the introduction of the paper, GAs have been widely used in search optimisation problems and with the aim to automate software testing.

Generally a GA consists of a group of individuals (population of genomes), each representing a potential solution to the problem in hand. An initial population with such individuals is usually selected at random. Then a parent selection process is used to pick a few of these individuals. New offspring individuals are produced using crossover, which keeps some of their parent's characterises and mutation, which introduces some new genetic material. An objective function, known as the *fitness function*, defines how close each individual is to being a solution and hence guides the search. The quality of each individual is hence measured by this fitness function, defined for the particular search problem. In our context, we consider the following notion.

Definition 9 The *fitness* is a function that given a TP of a TEFSM M, sums the penalty points (assigned through the transition ranking process for M and the temporal constrain ranking for M) for each of the transition of the TP.

In the next section we fully describe the fitness functions, used in two different contexts, that we consider in this paper.

Crossover exchanges information between two or more individuals. The mutation process randomly modifies offspring individuals. The population is iteratively recombined and mutated to evolve successive populations, known as generations. When the termination criterion specified is satisfied, the algorithm terminates.

When using GAs the first issue that needs to be addressed is how to represent potential solutions in the GA population. A genotype is how a potential solution is encoded in a GA, while the phenotype is the real representation of that individual. There are different representation techniques, the most common being binary and characters. Gray coding is a binary representation technique that uses slightly different encoding to standard binary. As it has been shown [55] that Gray codes are generally superior to standard binary by helping to

represent the solutions more evenly in the search space, we used it in this work.

The first step in a GA involves the initialisation of a population of usually randomly generated individuals. The size of the population is specified at the start. Every individual is evaluated using the fitness function. When ranking is used the population is sorted according to the fitness value of the individuals. Then each individual is ranked irrespective to the size of its and its predecessors fitness. This is known as linear ranking. It has been shown that using linear ranking helps reduce the chance of a few very fit individuals dominating the search leading to a premature convergence [39].

An important part of the algorithm is parent selection. A commonly used technique is the roulette-wheel selection. Here the chance of an individual being selected is directly proportional to its fitness or rank (if linear ranking is used). Hence the selection is biased towards fitter individuals.

The most common recombination technique used is crossover. During crossover the genes of the two parents are used to create one or more new offsprings. The simplest one is known as *single point crossover* [39]. In this work we also use single point crossover with a randomly generated crossover point.

Mutation is applied to each individual after crossover. It randomly alters one or more genes known as single point and multiple point mutation respectively [13]. A predefined mutation rate (typically the reciprocal of the chromosome length) determines which individuals are mutated. A single point mutation with randomly selected point has been also used [39].

There can be different termination criteria for a GA depending on the fitness function. If the fitness function is such that a solution would produce a specific fitness value, which is known, then the GA can terminate when an individual with such fitness is generated. However in many cases this is not known therefore the GA must be given other termination criteria. Such a criterion can be the specification of a maximum number of generations after which the GA will terminate irrespective of whether a solution has been generated. We use a combination of termination criteria.

4.2 Definition of the fitness function

In order to achieve our objectives we require an easy to compute fitness function that estimates the feasibility of a TP but also helps us test our temporal constraints. Computing the actual feasibility of a transition path is computationally expensive, so we need a method to estimate this.

Some transition paths consist of transitions with difficult to satisfy guards. It is always possible to execute a *simple* transition in a transition path since there are no guards to be satisfied. The presence of g_{P^i} transitions could render a transition path infeasible because of its input predicate guard. However the conditions of this guard are more likely to be satisfiable than domain guards because the values of the input parameters $P^i \subseteq P$ can be chosen. When these conditions depend also on some internal variables $V' \subseteq V$ then such g_{P^i} transitions might not be easier to trigger than g_D transitions. In some cases the execution of a g_D transitions could require reaching its start state through a specific transition path. The feasibility of g_{P^i}-g_D transitions depends on both issues outlined above for g_{P^i} transitions and g_D transitions.

Since the presence of g_D transitions and g_{P^i}-g_D transitions seem to increase the chance of a transition path being infeasible such transitions can be penalised and *simple* transitions rewarded in a TP. In this paper we take the feasibility estimation framework from previous work.

Secondly we may consider the temporal constraints of the transitions in a TP. For example a test sequence may be needed to stress test the temporal aspects of the system. Adequate test cases should be feasible (in order to be useful) and may focus on transitions with complex temporal constraints. However since the temporal constraints for every transition of M are dependant on the configuration of M (the current values if the internal variables) then it is difficult to know the exact temporal constraints without executing the system and verifying the configuration of M. If the temporal constraints for M are listed in a table then we can analyse the constraints and categorise different transitions in a similar way as we classified them according to their guards above. However if the temporal constraints are represented by one or more formulas the complexity of analysing all the possibilities the problem may become untrackable.

The idea of generating configuration confirming sequences has been already implemented [49]. The potential problem of combinatorial complexity associated with exploring all the configuration of an EFSM has been addressed by deriving a confirming test sequence for a designated reference configuration and a given black list of typical faulty configuration, supplied by the tester.

We may try to estimate the different temporal constraints associated with each transition according to how the temporal constraint is defined. Note that the same transition may have different temporal constraints depending on the values of the internal variables of M.

Some transitions may not have temporal constraints at all, while others might have fixed temporal constraints that are not dependant on the configuration of M. Other transitions may have temporal constraints that are expressed using tables while some may have the temporal constraints represented using formulas.

Based on these observations we may classify the transitions in a TEFSM M in the following way: *no-time transitions* are those transitions that have *no* temporal constraints; *fixed-time transitions* are those transitions that *have* temporal constraints that *are not* effected by the values of the internal variables V of M; *known-time transitions* are those transitions that *have* temporal constraints that *are* effected by the values of the internal variables V of M, but presented in an easy to analyse way; *variable-time transitions* are transitions that *have* temporal constraints that *are* effected by the values of the internal variables V of M, but are presented by one or more formulas and the temporal constraints are not easy to analyse without considering a subset of all the configurations of M.

A *transition ranking* process is completed before the fitness function can be used. This process first ranks each transition of the EFSM according to how many penalty points are assigned to the transition guards. A *simple* transition gets the highest rank (i.e. lowest amount of penalty points), an g_{P^i} transition is ranked next etc. Transitions that have the same number of penalty points get the same rank. This algorithm, in essence, sorts $|T|$ elements and, therefore, has complexity $O(|T|.log|T|)$ where $|T|$ is the number of transitions in M. Then the process ranks each transition according to the its temporal constraint category. In our case if we are attempting to stress test the implementation then we can argue that $variable-time$ transitions can potentially have the most complex temporal constrains hence be ranked highest (i.e. lowest amount of penalty points), $known-time$ transitions can be ranked next as they still dependant on the internal variables of M etc. The order can be reversed if the aim is to find a test sequence that will most likely work. Such test cases may be used in early stages of software development.

In this paper the two rankings are given equal weight before being combined, however different weights can be given to the rankings if required. We chose to give equal weight to the rankings following the conclusions of a similar experiment in [6] where different weights were used for a similar multi-optimisation problem. However further work can be done to consider different ways to combine the two matrices in the fitness function.

The fitness algorithm used in this work can be used to reward a potential solution to a TP generation problem according to the combined ranks of the transitions in the sequence. The fitness function reflects the belief that the fewer constraints a sequence contains, the more likely it is be feasible and the less we can analyse the temporal constraints of a transition, the more likely it is that they are more complex. It is important to note that our particular temporal constraints classification may not be fully applicable to all TEFSMs because it depends on the particular specifications. However the main principles should still hold even if different temporal constraints classification is used.

Estimating the feasibility of a TP is just the first part of the more difficult problem of generating actual IS for a TFTP that do not always represent the shortest path between two states. Also there may be other computationally inexpensive analysis of a TP that can be added to the existing fitness functions to make it more accurate. In this work we focus on evaluating our feasibility and complex temporal conditions estimation fitness function.

5 High performance computing and simulation

Due to the emerging trend of High Performance Scientific applications, techniques to validate these highly complex systems are specially demanded by the research community. This kind of applications is specially used in fields like astronomy [53], medicine [52] and earthquake modeling [1]. In general, these applications require a large amount of resources, like multicore CPUs and fast communication networks, to deal with very large data sets. In most cases the execution can take weeks and even months to be completed. If these applications are not correct, a lot of effort, time and resources are wasted. Consequently, it is necessary to provide mechanisms for validating, in particular through testing, applications before exploiting them. Major requirements for this kind of applications are scalability, reliability and feasibility. In High Performance Computing systems (HPC) the development of an implementation that satisfies a set of given requirements is a very difficult and complex task because there are many interrelated parameters that have an important influence in each one of these requirements.

Basically, there are two ways to perform studies of complex HPC environments. The first method consists in running the desired application in a real hardware-based system and then measuring the performance obtained. The second method consists in running the same application, or a simplified version of it, in a simulated environment representing the real system. Both methods can be used to analyze and predict the performance and behavior of different applications. However, simu-

lation methods have their own advantages and disadvantages. Some of them are:

- Simulation experiments are less expensive and more flexible than hardware-based experiments because they do not require modifying the real system to analyze different possibilities.
- Simulation experiments can be launched on any hardware platform.
- In many cases, simulation experiments are more time-expensive than hardware-based experiments. This problem can be minimized by adding hardware resources for the simulation, for example, parallelizing the simulation execution on a huge computing cluster.
- Results obtained from simulation need to be va0lidated to ensure their accuracy.
- Scaling the architecture of the real system is more expensive and time-consuming that performing the same changes in a simulated environment.
- Simulators can be shared easily with other researchers, while hardware is more difficult to share.
- Simulation only takes care of these aspects we have included on it. Therefore, the possibility that one element not included results to be the key of the performance is always there.

In this work we use the SIMCAN simulation platform [44] to modelling, simulating and testing a High Performance Scientific application called BIPS3D [30]. Our results are described in the next section. The SIMCAN simulation platform is oriented towards the simulation of different kinds of parallel and distributed systems. SIMCAN has been designed to provide flexibility, accuracy, performance and scalability, which makes it a powerful simulation platform for designing, testing and analyzing both actual and possible architectures. The range of systems to simulate covers from a single computing node to a complete high performance distributed system. The best feature of SIMCAN is its ability to model and simulate large environments (thousands of nodes) with a customizable level of detail. The way SIMCAN performs a simulation depends both on the user requirements and on the resources available. Therefore, SIMCAN can be either executed in a single computer using sequential simulation or it can be executed in parallel using both shared memory computers and distributed memory systems. The speed of the simulation will depend highly of the computing resources used for executing the simulation. The more CPU and memory resources available, the better performance will be obtained for executing the simulation.

The current version of SIMCAN provides a wide set of developed components in order to build simulated environments. Although using those components a great variety of architectures can be modeled and simulated, the design of this simulation platform let users add its own new components to the repository of SIMCAN. Thus, new environments with more specific configurations can be built. Moreover, SIMCAN offers different programming schemas and a complete system API with configurable facilities, which let build any application model from scratch. Those application models could be implemented using statistical approaches or could be implemented as a port of the real application with more or less detail.

6 Case studies

In this section we present two different case studies, which consider temporal constraints of a certain class of a finite state machines. Thus, the previously described GAs and fitness function are applied to these scenarios. Each case study shows how the test case generation problem can be presented as a search problem and automated. In both cases, different GAs techniques are compared with random search to evaluate the proposed approach.

In order to check the usability and scalability of this approach, two scenarios with different scale sizes are evaluated. First, a basic communication protocol is analyzed. Second, a real application, called BIPS3D, has been modeled and evaluated using a simulation platform for modeling and simulating distributed systems and applications. While the first case study shows in a simple way, the main features of our methodology, the second case study shows that our framework can cope with complex systems. In fact, this case study illustrates how different research lines such as formal methods, artificial intelligence, and simulation can be successfully combined to solve difficult problems.

6.1 Class 2 transport protocol

The Class 2 transport protocol, in the following M_1, is presented in Figure 1 and the corresponding transition table (excluding the conditions and temporal constraints) is shown in Table 1. This table also shows the ranked transition table for M_1. For example t_3 and t_{10} share the same temporal constraint classification and therefore are ranked lower than some other transitions however they have different feasibility ranking due to the differently classified guards they have.

The search for a TP that is likely to be feasible and yet have complex temporal constraints is represented as a fitness minimisation problem. The GA is then used

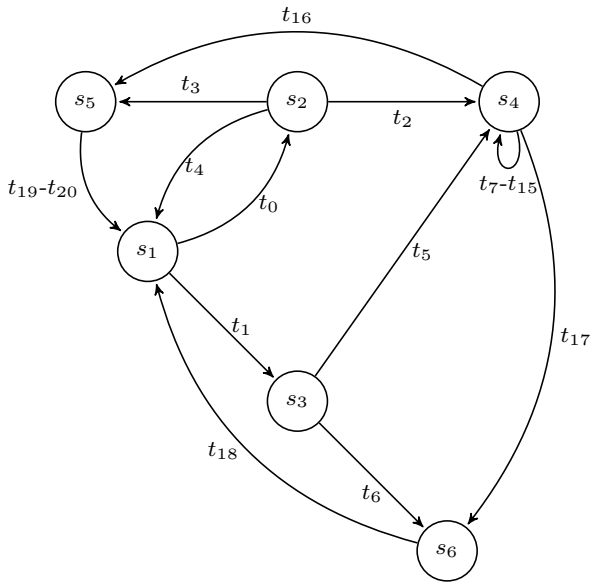

Fig. 1 Class 2 transport protocol TEFSM M_1. The transition table is on Table 1.

Table 1 Temporal constraint ranking and feasibility ranking for all transitions in M_1

t	input	output	feasibility rank	temporal rank
t_0	ICONreq	!CR	0	0
t_1	CC	!ICONconf	0	0
t_2	T_expired	!CR	2	0
t_3	T_expired	!IDISind	1	1
t_4	IDATreq	DT	0	0
t_5	AK		6	1
t_6	AK		6	1
t_7	AK	DT	5	0
t_8	AK	!IDISind	4	0
t_9	T_expired	DT	3	0
t_{10}	T_expired	!IDISind	2	1
t_{11}	DR	!IDISind	0	2
t_{12}	DR	!IDISind	0	2
t_{13}	DR	!IDISind	0	2
t_{14}	DR	!IDISind	0	2

to search for appropriate solutions. The same computational effort is also used with a random TP generator using the same fitness function and result verification as the GA. This search problem uses a fitness function that rewards transition sequences with higher ranked transitions and penalises invalid transitions. It produces a numerical value potentially showing how close an input sequence is to defining a valid TFTP. The fitness function represents the search for a TFTP sequence as a function minimisation problem so an AIS with a lower fitness value is considered to be more likely to form a TFTP since it is made up of more highly ranked transitions.

The fitness does not guarantee that a particular transition path can be triggered or that it contains the most complex temporal constraints in M. It makes sure that it is constructed using consecutive transitions that are highly ranked. The verification process then checks if an IS can be generated to trigger such a TP. The verification method is similar to previous work [6] where a TP is evaluated by resetting M to its initial configuration and attempted to be trigger the TP in our simulated implementation. The process is repeated several times and the overall result of how many times the TP was correctly triggered are counted and compared to the times it failed. Hence an estimation is derived to measure the feasibility of these TPs.

In our example we looked at a relatively simple temporal constraints range (hence the small range of rankings on Table 1) and it was easy to manually check the complexity of the temporal constraints for each transition. This was sufficient for our case study, but defining an automated estimation measure for the temporal qualities of a TP remains future work.

In order to compare the performance of the GA and Random algorithms TFTP generation two different metrics are used. *State coverage* is the number of cases where at least one TFTP was generated for every TFTP size attempted from each state in M and *success rate* is the number of TFTPs that were generated compared to the total number of attempts it took to generate the results.

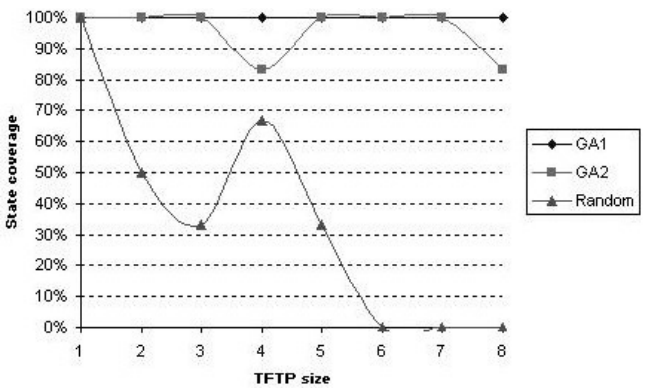

Fig. 2 State coverage for PB notation TFTPs generated using GA and Random generation algorithms for M_1 with 1-8 transitions

In this case study two slightly different GAs were used to compare their performance when applied to this problem. The first GA used a single point crossover and mutation while the second used a complex multiple point crossover and mutation. In general the second GA tended to find a solution slightly faster than the first GA, but they produced the same results.

Table 2 represents a summary of the result averages. In general the results show that the GAs seem

to perform better than the Random generation method according to both metrics. Figure 2 represents the state coverage results for all the different TFTP sizes. GA1 performs well and GA2 fails to find only one TFTP of size 4 and one of size 8, while the random generation algorithm performance peaks when generating TFTPs of size 4 and declines as the TFTP size increases. Clearly the GAs outperform the Random generation method for TFTPs of more than one transition. Figure 3 represents the success rate results for all the different TFTP sizes in our case study. The high fluctuation here can be explained by the different degree of difficulty in generating TFTPs of different sizes for some states. This relates to the guards of transition $t14$, which is one of the core transitions. Although the guard conditions are complex in the context of M_1 they are very easy to satisfy. Hence the random algorithm can easily select $t14$ in its TP search, while the GAs try to avoid it without realising that it should not. GA1 performs mostly better than the random generation algorithm, except for TFTPs of size 4 (average performance of 54%). For TFTPs of size 4 the random algorithm performs slightly better than GA1 and GA2. The random generation method did not find any TSTPs for sizes 4 to 6 while the GAs have different success rate for each size. This shows how different states have different properties. Hence future work may focus on more analysis of the guards and the temporal conditions to even out the search performance.

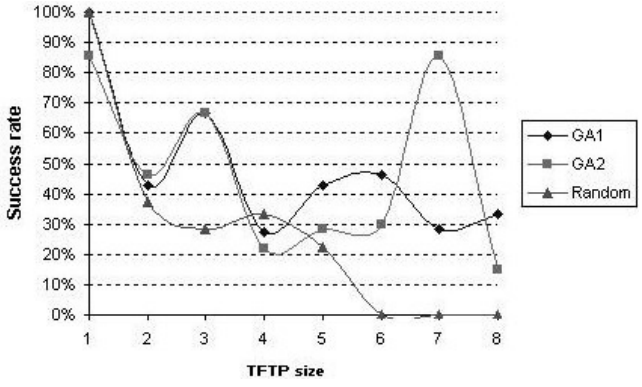

Fig. 3 Success rate for PB notation TFTPs generated using GA and Random generation algorithms for M_1 with 1-8 transitions

For both metrics the two GA search algorithms perform on average better than the random generation algorithm. This suggests that the fitness function here helps guide a heuristic search for TFTPs. In Figure 2 and Figure 3 we observe that as longer TFTPs (and possibly when larger TEFSMs) are considered, the heuristics seems to perform increasingly better than the random generation algorithm when given equal processing effort in terms of fitness evaluations and TFTP verifications. On all occasions the TFTPs generated by the GAs the had equivalent or more complex temporal constraints compared to those generated using the random TP generation method. For a TEFSM of this size, as in our case study, it is expected to have similar performance for the small TFTPs because the search space in those situations is not that big. However as the search space is increased (in or case study by increasing the size of the TFTPs) it becomes clear that a random generation approach finds it hard to generate TPs that feasible and satisfy the temporal constraints.

The state coverage metric is the easier one to satisfy. Not surprisingly the GAs found at least one TFTP for every state in M_1. This measure however discards all the unsuccessful attempts to generate a given TFTP. Hence the success rate metric considers those unsuccessful attempts as well. The success rates results are lower but the GAs seem to outperform the random algorithm.

Table 2 GA and Random search result averages for the Class 2 protocols for TFTPs with 1-8 transitions.

	State Coverage	Success rate
GA1	100%	48%
GA2	96%	47%
Random	35%	28%

Overall both GAs performed well and generated very similar results. This indicates that the fitness function and the TP representation represent the problem of TFTP generation reasonably well.

6.2 The BIPS3D application

BIPS3D [30] is a scientific application that performs 3-dimensional simulations of BJT and HBT bipolar devices. The goal of the 3D simulation is to relate electrical characteristics of the device with their physical and geometrical parameters. The basic equations to be solved are Poisson's equation and electron and hole continuity, in a stationary state.

Figure 4 shows a basic schema of the different phases of the BIPS3D application. This application starts with a set of physics parameters belonging to the materials that compose a transistor. In addition, we need an unstructured mesh in which we place more nodes in the areas of union between different areas of the transistor. The mesh is split by the master process into sub-domains using the METIS library [26]. Each one

of these sub-domains is sent to one process (slave) to compute it. When each of the slaves finishes to process the corresponding sub-domain, generated data must be sent to the master process to merge all resulting data corresponding to the rest of processes. Finally, the results are written to a file.

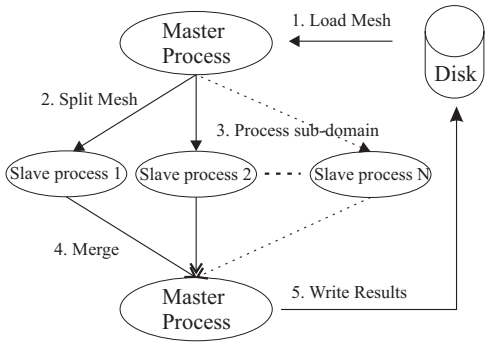

Fig. 4 Basic schema of BIPS3D

The TEFSM M_2 that models the behavior of BIPS3D is presented in Figure 5 and the corresponding transition table (excluding the conditions and temporal constraints) is shown in Table 3.

In this case study, we have used the SIMCAN simulation platform [44] to model the BIPS3D application. SIMCAN is a fast, flexible, scalable and expandable simulation platform for modeling and simulating distributed systems. The main principle of SIMCAN lies on integrating the model of the four basic systems into a single simulation platform: CPU, memory, network and storage. Since SIMCAN offers the event-programming paradigm and a complete API with configurable facilities, the applications that can be represented using TEFSMs fit especially well in this simulation platform. The event-programming paradigm is the normal way to develop applications and modules in SIMCAN. This paradigm is specially fitted for programming simulations that have to deal with all the events that both the hardware and the software produce. Applications developed using the event-programming paradigm are especially useful when TEFSMs are used to specify the system.

The BIPS3D application has been modeled using SIMCAN in the past. In these previous cases, the generated models were used to study the performance of BIPS3D in distributed systems using different architectural configurations [46,45]. However, in this paper the model M_2 is used to generate test sequences for temporally constrained systems and, in particular, in generating timed feasible transition paths (TFTPs) with specific properties that can in turn be used to generate test input. From the simulator's point of view, the application's execution consists of a set of stages (transitions). The execution of all stages represents one iteration (TP) and it must be performed in order, starting from the synchronization stage and finishing at the results stage.

Table 3 Temporal constraint ranking and feasibility ranking for all transitions in M_2

t	input	output	feasibility rank	temporal rank
t_0	iDat	fDat	0	1
t_1	eDat	WTG	2	0
t_2	SYNC	S_STBL	5	5
t_3	iPSD	Err_WTG	0	0
t_4	iSD	fSD	5	3
t_5	iMD	fMD	4	0
t_6	RTR		0	0
t_7	iPMD	fPMD	2	2
t_8	RTR		2	1
t_9	eMD	WTG	0	2
t_{10}	iSD	fSD	5	0
t_{11}	AK		6	5
t_{12}	iSD	fSD	0	0
t_{13}	iPSD	fPSD	0	0
t_{14}	iRST	fRST	6	0
t_{15}	eSD	RTR	0	4
t_{16}	iPSD	fPSD	2	0
t_{17}	iRST	fRST	2	0
t_{18}	T_expired	RTR	2	4
t_{19}	T_expired	RTR	0	0
t_{20}	T_expired	RTR	1	0
t_{21}	iRST	ErrWTG	1	2
t_{22}	iRST	ErrWTG	2	0
t_{23}	iRST	ErrWTG	1	0
t_{24}	RTR		0	5

The event-programming that we have mentioned before has some advantages. The most important one is scalability. Thus, scaling the model represented by a given graph is immediate, because it is enough to configure the corresponding parameters like the number of processes, and the amount of data processed in each operation. Another advantage is that we can study the impact of changes in the application on the overall system performance, without changing any line of code. It can be done simply by configuring the corresponding parameters with the right values.

Thus, the SIMCAN simulation platform has been used to model the BIPS3D application and the underlying hardware to execute it. One of the best advantages of using simulation, instead of executing the real application in a real system, is that we can build a virtual environment and initializing it in a specific state to execute a given application model. This capability of customizing and initializing the execution of a modeled application let us to study and reproduce TPs easily. This can be especially relevant in those environments that contain shared resources, where it is necessary to

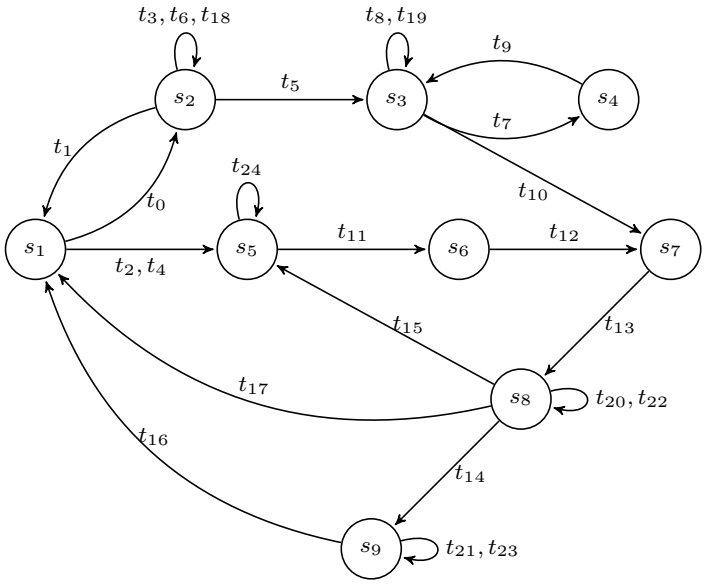

Fig. 5 BIPS3D applicaiton TEFSM M_2. The transition table is on Table 3.

consider the correctness of the system results as well as specific time requirements. Consequently, the TEFSM shown in Figure 5 has been programmed in the SIMCAN simulation platform. In such a way that we can apply the techniques described in this paper, by using GAs and a fitness function, to generate and analyze test cases for testing a corresponding application.

Similar to the situation in the previous case study, the fitness function used in this case rewards transition sequences with higher ranked transitions and impose a penalty to invalid transitions. Table 3 shows the initial values computed by the initialization phase before applying the fitness function. However, the fitness function used in this scenario is more sophisticated than the function used in the previous case study. In this case, the weight assigned to each rank varies dynamically. These weights are computed by the simulation kernel depending on the penalty points assigned to the transition guards. Thus, the simulation kernel ranks each weight according to the temporal constraint. Therefore, the transitions that can potentially have the most complex temporal constrains are given a higher weight. The main objective of the fitness function is to construct a TP by using consecutive highly ranked transitions. The feasibility of the created TPs is measured by comparing how many times the TP was correctly triggered with respect to the times it failed.

Due to the complexity of generating test sequences for distributed applications, in this case study we merge the advantages of three different fields to aid test sequence generation: formal testing, artificial intelligence and simulations. Thus, apart from modeling the behavior of BIPS3D in SIMCAN, a module that contains a new engine for running GAs has been developed. Therefore, the process to create test cases using genetic algorithms and simulations is totally automated. This module contains the logic to apply the two techniques described in the previous case study: single point crossover with mutation (GA1) and complex multiple point crossover with mutation (GA2). Both techniques use the same fitness function. Also, a random search has been used to compare the performance obtained in each GA.

Figures 6 and 7 show the performance of the GA and Random search using two metrics, *state coverage* and *success rate*, respectively. Table 4 presents a summary of the result averages. This table shows clearly that random generations provides poor results in both metrics, and G1 is slightly better than G2.

Figure 6 represents the state coverage results for all the different TFTP sizes. This chart shows that GA1 performs well, while GA2 fails to find some TFTP's for specific sizes, such as 4, 8, 12 and from 14 to 16. However, random search provides a coverage of 50% for TFTP's of size 4, and consequent results degenerate as the size increases. Clearly the Random generation method is not a feasible solution, because the state coverage for TFTP's of size greater than 1 is not acceptable. In contrast, GAs provide good results by covering in the worst case 75% of the states.

Figure 7 represents the success rate results for all the different TFTP sizes in this case study. As occurs with the previous case study, in this case we obtain high fluctuation in success rate for both GAs. Similarly, it can

be explained by the different degree of difficulty in generating TFTPs of different sizes for some states, which is more emphasized in this case because we use double of states, 16 instead of 8. Moreover, in this case study there are some core transitions with complex guard conditions, such as t_{15} and t_4.

This chart (see Figure 7) shows that success rate using random search drops quickly. It is mainly caused when a core transition is selected, which is in most cases, difficult to satisfy. The only case when random search is better than GAs is for TFTP's of size 3. In the rest of the cases, random search provides significantly worst results than GA. In fact, as the search space is increased (using sizes greater than 7) random approach can barely generate feasible TPs that satisfy the temporal constraints. Otherwise, when comparing GA1 with GA2, we can observe that in average GA1 provides slightly better results than G2. There are only three cases where G2 provides better results than G1 (TFTP's of size 8, 9 and 11). Using TFTP's of size greater than 2, the best results provided by GA1 is 60%, while the best results provided by GA2 is 65%. Otherwise, the worst case is the same for both algorithms, 20%.

In general, as in the previous case study, GAs algorithms obtains better results than the random generation algorithm on both metrics. Also, it is important to mention that in this case study the results are very similar to the results obtained in the previous one. In this case, we use TFTP's of size 16, instead of 8. This means that our fitness function guides well the GA to find a solution. Thus, performing simulations of the execution is very useful to compute the specific state and its corresponding condition guards. In Figure 6 and Figure 7 we can observe that as longer TFTPs (and possibly when larger TEFSMs) are considered, the random search does not scale at all, while GAs maintain a balance in the performance.

Table 4 GA and Random search result averages for BIPS3D for TFTPs with 1-16 transitions.

	State Coverage	Success rate
GA1	100%	47.5%
GA2	94.0%	41,3%
Random	21.5%	15.9%

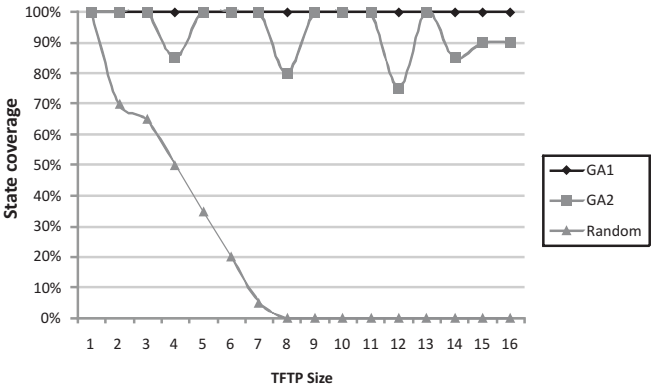

Fig. 6 State coverage for PB notation TFTPs generated using GA and Random generation algorithms for M_2 with 1-16 transitions

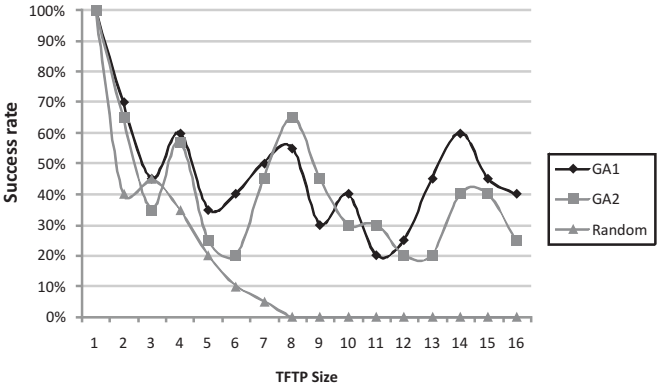

Fig. 7 Success rate for PB notation TFTPs generated using GA and Random generation algorithms for M_2 with 1-16 transitions

7 Conclusions and Future work

In this paper we have presented a complete framework that provides a computationally inexpensive method to address the important problem of test data generation for TEFSMs. This work extends previous work on feasibility of EFSMs [8,9] to consider temporal constraints.

We defined the problem of finding transition sequences that are likely to be feasible, and to satisfy some temporal criteria, as a search problem. We defined a computationally efficient fitness function that is used to guide our GAs. We have considered two case studies to evaluate the goodness of our methodology. We have that the GAs almost fully satisfy our coverage criteria and increasingly outperform random generation as the TFTP size increases. Even though the success rate fluctuated, the average success rate of the GAs was almost double than that of the randomly generated results. Overall, our results suggest that the approach scales well and can be applied to even larger TEFSMs.

Future work may focus on refining the fitness function to take into account loops and other difficulties to estimate transitions. Further analysis of the conditions might also help. A more thorough evaluation of the fitness function on other TEFSMs may also be beneficial to evaluate further how well the method scales. Dif-

ferent TEFSMs may present the need for alternative temporal constraint classification, which will be very interesting to investigate.

Acknowledgements We would like to thank Karnig Derderian for his participation in the previous stages of this research.

References

1. Akcelik, V., Bielak, J., Biros, G., Epanomeritakis, I., Fernandez, A., Ghattas, O., Kim, E.J., Lopez, J., O'Hallaron, D., Tu, T., Urbanic, J.: High resolution forward and inverse earthquake modeling on terascale computers. In: 16th ACM/IEEE conference on Supercomputing, SC'03. ACM Press (2003)
2. Ammann, P., Offutt, J.: Introduction to Software Testing. Cambridge University Press (2008)
3. Blanquart, J.P., Armengaud, E., Baufreton, P., Bourrouilh, Q., Griessnig, G., Krammer, M., Laurent, O., Machrouh, J., Peikenkamp, T., Schindler, C., Wien, T.: Towards cross-domains model-based safety process, methods and tools for critical embedded systems: The CESAR approach. In: 30th Int. Conf. on Computer Safety, Reliability, and Security, SAFECOMP'11, LNCS 6894, pp. 57–70. Springer (2011)
4. Bosik, B.S., Uyar, M.Ü.: Finite state machine based formal methods in protocol conformance testing. Computer Networks & ISDN Systems **22**, 7–33 (1991)
5. Chow, T.: Testing software design modeled by finite state machines. IEEE Transactions on Software Engineering **4**, 178–187 (1978)
6. Derderian, K.: Automated test sequence generation for finite state machines using genetic algorithms. Ph.D. thesis, Brunel University (2006)
7. Derderian, K., Hierons, R.M., Harman, M., Guo, Q.: Input sequence generation for testing of communicating finite state machines (CFSMs). In: 6th Annual Conf. on Genetic and Evolutionary Computation, GECCO'04, LNCS 3103, pp. 1429–1430. Springer (2004)
8. Derderian, K., Hierons, R.M., Harman, M., Guo, Q.: Automated Unique Input Output sequence generation for conformance testing of FSMs. Computer Journal **49**(3), 331–344 (2006)
9. Derderian, K., Hierons, R.M., Harman, M., Guo, Q.: Estimating the feasibility of transition paths in extended finite state machines. Automated Software Engineering **17**(1), 33–56 (2010)
10. Derderian, K., Merayo, M.G., Hierons, R.M., Núñez, M.: Aiding test case generation in temporally constrained state based systems using genetic algorithms. In: 10th Int. Conf. on Artificial Neural Networks, IWANN'09, LNCS 5517, pp. 327–334. Springer (2009)
11. Derderian, K., Merayo, M.G., Hierons, R.M., Núñez, M.: A case study on the use of genetic algorithms to generate test cases for temporal systems. In: 11th Int. Conf. on Artificial Neural Networks, IWANN'11, LNCS 6692, pp. 396–403. Springer (2011)
12. Duale, A.Y., Uyar, M.Ü.: A method enabling feasible conformance test sequence generation for EFSM models. IEEE Transactions on Computers **53**(5), 614–627 (2004)
13. Goldberg, D.: Genetic Algorithms in Search, Optimisation and Machine Learning. Addison-Wesley (1989)
14. Grieskamp, W., Kicillof, N., Stobie, K., Braberman, V.: Model-based quality assurance of protocol documentation: tools and methodology. Software Testing, Verification and Reliability **21**(1), 55–71 (2011)
15. Guo, Q., Hierons, R.M., Harman, M., Derderian, K.: Computing Unique Input/Output sequences using genetic algorithms. In: 3rd Int. Workshop on Formal Approaches to Testing of Software, FATES'03, LNCS 2931, pp. 164–177. Springer (2003)
16. Harman, M., McMinn, P.: A theoretical and empirical study of search-based testing: Local, global, and hybrid search. IEEE Transactions on Software Engineering **36**(2), 226–247 (2010)
17. Hennie, F.: Fault-detecting experiments for sequential circuits. In: 5th Annual Symposium on Switching Circuit Theory and Logical Design, pp. 95–110 (1964)
18. Hierons, R.M., Bogdanov, K., Bowen, J., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luettgen, G., Simons, A., Vilkomir, S., Woodward, M., Zedan, H.: Using formal methods to support testing. ACM Computing Surveys **41**(2) (2009)
19. Hierons, R.M., Bowen, J., Harman, M. (eds.): Formal Methods and Testing, LNCS 4949. Springer (2008)
20. Hierons, R.M., Merayo, M.G., Núñez, M.: Testing from a stochastic timed system with a fault model. Journal of Logic and Algebraic Programming **78**(2), 98–115 (2009)
21. Hierons, R.M., Ural, H.: Reduced length checking sequences. IEEE Transactions on Computers **51**(9), 1111–1117 (2002)
22. Hierons, R.M., Ural, H.: Optimizing the length of checking sequences. IEEE Transactions on Computers **55**(5), 618–629 (2006)
23. Jones, B.F., Eyres, D.E., Sthamer, H.H.: A strategy for using genetic algorithms to automate branch and fault-based testing. The Computer Journal **41**(2), 98–107 (1998)
24. Kalaji, A.S., Hierons, R.M., Swift, S.: Generating feasible transition paths for testing from an extended finite state machine (EFSM). In: 2nd Int. Conf. on Software Testing Verification and Validation, ICST'09, pp. 230–239. IEEE Computer Society (2009)
25. Kalaji, A.S., Hierons, R.M., Swift, S.: An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models. Information & Software Technology **53**(12), 1297–1318 (2011)
26. Karypis, G.: METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices. version 5.0. Tech. rep., Department of Computer Science & Engineering, University of Minnesota (2011)
27. Laurent, O.: Using formal methods and testability concepts in the avionics systems validation and verification (V&V) process. In: 3rd Int. Conf. on Software Testing, Verification, and Validation, ICST'10, pp. 1–10. IEEE Computer Society (2010)
28. Lee, D., Yannakakis, M.: Testing finite state machines: State identification and verification. IEEE Transactions on Computers **43**, 306–320 (1994)
29. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines: A survey. Proceedings of the IEEE **84**(8), 1090–1123 (1996)
30. Loureiro, A., González, J., Pena, T.F.: A parallel 3D semiconductor device simulator for gradual heterojunction bipolar transistors. Journal of Numerical Modelling: Electronic Networks, Devices and Fields **16**(1), 53–66 (2003)

31. Mairhofer, S., Feldt, R., Torkar, R.: Search-based software testing and test data generation for a dynamic programming language. In: 13th Annual Conf. on Genetic and Evolutionary Computation, GECCO'11, pp. 1859–1866. ACM Press (2011)
32. McMinn, P.: Search-based software test data generation: a survey. Software Testing Verification and Reliability **14**(2), 105–156 (2004)
33. Merayo, M.G., Núñez, M., Hierons, R.M.: Testing timed systems modeled by stream X-machines. Software and Systems Modeling **10**(2), 201–217 (2011)
34. Merayo, M.G., Núñez, M., Rodríguez, I.: Generation of optimal finite test suites for timed systems. In: 1st IEEE & IFIP Int. Symposium on Theoretical Aspects of Software Engineering, TASE'07, pp. 149–158. IEEE Computer Society Pres (2007)
35. Merayo, M.G., Núñez, M., Rodríguez, I.: Extending EFSMs to specify and test timed systems with action durations and timeouts. IEEE Transactions on Computers **57**(6), 835–848 (2008)
36. Merayo, M.G., Núñez, M., Rodríguez, I.: Formal testing from timed finite state machines. Computer Networks **52**(2), 432–460 (2008)
37. Merayo, M.G., Núñez, M., Rodríguez, I.: A formal framework to test soft and hard deadlines in timed systems. Software Testing, Verification and Reliability (2012). Accepted for publication, doi: 10.1002/stvr.448
38. Michael, C.C., McGraw, G., Schatz, M.A.: Generating software test data by evolution. IEEE Transactions on Software Engineering **27**(12), 1085–1110 (2001)
39. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd, revised and extended edn. Springer (1996)
40. Molinero, C., Núñez, M., Andrés, C.: Combining genetic algorithms and mutation testing to generate test sequences. In: 10th Int. Conf. on Artificial Neural Networks, IWANN'09, LNCS 5517, pp. 343–350. Springer (2009)
41. Molinero, C., Núñez, M., Hierons, R.M.: Creating adaptive sequences with genetic algorithms to reach a certain state in a non-deterministic FSM. In: IEEE Symposium on Artificial Life, ALIFE'11, pp. 22–29. IEEE Computer Society (2011)
42. Molinero, C., Núñez, M., Hierons, R.M.: Experimental comparison of different techniques to generate adaptive sequences. In: 11th Int. Conf. on Artificial Neural Networks, IWANN'11, LNCS 6692, pp. 404–411. Springer (2011)
43. Myers, G.: The Art of Software Testing, 2nd edn. John Wiley and Sons (2004)
44. Núñez, A., Fernández, J., Filgueira, R., García, F., Carretero, J.: SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications. Simulation Modelling Practice and Theory **20**(1), 12–32 (2012)
45. Núñez, A., Fernández, J., García, J.D., Carretero, J.: Analyzing scalable high-performance I/O architectures. In: 3rd Int. Conf. on Parallel and Distributed Processing Techniques and Applications, PDPTA'08, pp. 631–637 (2008)
46. Núñez, A., Fernández, J., García, J.D., García, F., Carretero, J.: New techniques for simulating high performance MPI applications on large storage networks. The Journal of Supercomputing **51**(1), 40–57 (2010)
47. Oh, J., Harman, M., Yoo, S.: Transition coverage testing for simulink/stateflow models using messy genetic algorithms. In: 13th Annual Conf. on Genetic and Evolutionary Computation, GECCO'11, pp. 1851–1858. ACM Press (2011)
48. Petrenko, A.: Fault model-driven test derivation from finite state models: Annotated bibliography. In: 4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067, pp. 196–205. Springer (2001)
49. Petrenko, A., Boroday, S., Groz, R.: Confirming configurations in EFSM testing. IEEE Transactions on Software Engineering **30**(1), 29–42 (2004)
50. Ramalingom, T., Thulasiraman, K., Das, A.: Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines. Computer Communications **26**(14), 1622–1633 (2003)
51. Srinivas, M., Patnaik, L.M.: Genetic algorithms: A survey. IEEE Computer **27**, 17–27 (1994)
52. Stone, S.S., Haldar, J.P., Tsao, S.C., Hwu, W.W., Liang, Z.P., Sutton, B.P.: Accelerating advanced MRI reconstructions on GPUs. In: 5th Conference on Computing Frontiers, CF'08, pp. 261–272. ACM Press (2008)
53. Szalay, A.S., Kunszt, P.Z., Thakar, A., Gray, J., Slutz, D., Brunner, R.J.: Designing and mining multi-terabyte astronomy archives: the sloan digital sky survey. In: 26th ACM SIGMOD Int. Conf. on Management of data, pp. 451–462. ACM Press (2000)
54. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann (2007)
55. Whitley, D.: A free lunch proof for gray versus binary encodings. In: 1st Genetic and Evolutionary Computation Conference, GECCO'99, pp. 726–733. Morgan Kaufmann (1999)