A Multi-Objective Evolutionary Algorithm to Obtain Test Cases With Variable Lengths

Thaise Yano, Eliane Martins
Institute of Computing
State University of Campinas
Campinas, SP, Brazil
{tyano,eliane}@ic.unicamp.br

Fabiano Luis de Sousa
Space Mechanics and Control Division
National Institute for Space Research
São José dos Campos, SP, Brazil
fabiano@dem.inpe.br

ABSTRACT

In this paper a new multi-objective implementation of the generalized extremal optimization (GEO) algorithm, named M-GEO$_{vsl}$, is presented. It was developed primarily to be used as a test case generator to find transition paths from extended finite state machines (EFSM), taking into account not only the transition to be covered but also the minimization of the test length. M-GEO$_{vsl}$ has the capability to deal with strings whose number of elements vary dynamically, making it possible to generate solutions with different lengths. The steps of the algorithm are described for a general multi-objective problem in which the solution length is an element to be optimized. Experiments were performed to generate test case from EFSM benchmark models using M-GEO$_{vsl}$ and the approach was compared with a related work.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Algorithms, Experimentation, Verification

Keywords

Multi-objective evolutionary algorithm, model-based testing, variable length test case

1. INTRODUCTION

Although most of search-based software engineering works is concerned with testing [11], few approaches have been proposed for model-based testing (MBT). The test cases are derived from the system behavior models in MBT. Models can help to understand the system, can adapt well to changes in the system and also can be independent of programming languages and platforms. Testing from state models, as extended finite state machine (EFSM), generally consists in generating transition path (i.e. test case) based on a coverage criterion (e.g. all transitions) and also in generating adequate data for the parameters to take the path. Since determining the test data is an undecidable problem, search-based software testing (SBST) can be a promising direction to systematically generate test cases from EFSM. In contrast to code-based testing in SBST, a crucial difference in MBT regards the number of design variables, i.e., elements to be optimized. The code-based testing searches for the input values of the program under test to cover the test criterion, then the number of design variables is known. In MBT, a test case consists of a sequence of inputs, corresponding to a transition path in the model, and its length is also of concern to test case generation. In the MBT approaches based on search-based techniques, the test case length is established a priori and does not change during the evolution process. Kalaji et al. [13, 12] used a genetic algorithm to generate paths from EFSM with fixed length and, after the search, data were obtained to sensitize the paths. McMinn and Holcombe [18] proposed a SBST approach for the generation of sequence of function calls with their parameters using chaining approach to represent the sequence but also predefined the sequence length. Lefticaru and Ipate [15] applied search techniques to find the parameters values of a function calls sequence represented as a path in a state machine. As they chose the path to be covered, the sequence length and the number of parameters were known.

Determining the sequence length automatically motivated us to develop the evolutionary algorithm called M-GEO$_{vsl}$ (Multi-Objective Generalized Extremal Optimization with variable string length) that could optimize the sequence length according to the fitness criterion. The algorithm is derived from a multi-objective version of GEO (Generalized Extremal Optimization) [7], introducing the capability to generate solutions with different lengths during the evolution process. As a multi-objective approach, M-GEO$_{vsl}$ is aimed at solving problems when the solutions need to meet several conflicting objectives simultaneously. In the context of SBST, GEO methods have been successfully used to automatically generate test cases [1, 21, 22, 5]. M-GEO$_{vsl}$ was briefly described in previous works [21, 22] addressing only to test case generation. In this paper, we present the generic steps for the implementation of the algorithm to be applied in a multi-objective problem where the solution length is itself an element to be optimized.

Using M-GEO$_{vsl}$, we propose an approach that treats the test case generation from EFSM as a multi-objective optimization problem, which simultaneously finds the path that contains a required transition and also minimizes the test sequence length. The paths as well as all the data involved in the path are generated during evolution process, including the parameter values. An open problem of testing from EFSM is the infeasible path generation, since conflicts can exist in guard conditions along a path. This problem is avoided, at least at the model level, by obtaining the path dynamically instead of analyzing only the structure of the model as usual. The approach has been evaluated for various benchmark models, as well as for real world applications, which shows its feasibility. In this paper, the experimental results for some benchmark models are presented. Moreover, the solutions generated in our approach is compared to the results of a related work [13]. In relation to multi-objective optimization for software testing, the approaches focus on different purposes: branch coverage and dynamic memory consumption of programs [14], coverage, cost and fault history for regression testing [23].

The paper is organized as follows. Section 2 presents M-GEO$_{vsl}$. Section 3 describes the test case generation approach from EFSM using M-GEO$_{vsl}$ and shows the results of the experiments. Section 4 concludes.

2. M-GEO$_{VSL}$

This section describes the main aspects of M-GEO$_{vsl}$.

2.1 Solution representation

M-GEO (Multi-Objective Generalized Extremal Optimization) [10] is a multi-objective implementation of GEO [7] based on the Pareto optimality. GEO algorithm is a global search meta-heuristic that generalizes the extremal optimization (EO) method [4] inspired by a model of natural evolution. GEO is specially devised to be used in constrained or unconstrained problems, non-convex or even disjoint design spaces, with any combination of continuous, discrete or integer variables [8, 9].

In GEO methods, the population consists of a string of species and for each of them is associated a fitness value. A first difference between M-GEO and M-GEO$_{vsl}$ is the population encoding. Each design variable in M-GEO is represented by bits and each bit is considered a species. In M-GEO$_{vsl}$ a discrete encoding is used, in which each design variable is treated as a species. For instance, in Figure 1, the population with 6 design variables has 6 species in M-GEO$_{vsl}$, while in M-GEO has 18 species, if each design variable is encoded with 3 bits. The main differential of M-GEO$_{vsl}$ is its capability to cope with different population lengths during the evolution process. For this purpose, the population consists of two parts (sub-strings): one with variable size and another with fixed size . In the variable part, the first species determines the length of the variable sub-string in a given moment of the evolution process. The fixed size part of the string can be optional and takes into account design variables of the problem that do not need a dynamic size representation.

The evolution process of M-GEO$_{vsl}$ (Section 2.3) is similar to M-GEO, however the particularities of each algorithm influence the steps of the process in relation to the population encoding and mutation operator (Section 2.2).

2.2 Mutation operator

Each species is mutated according to its domain. A special case is the mutation of the first species of the population's variable part. Two situations should be considered when this species is mutated: the new value increases or decreases the current one. In the first case, new species are appended to the end of the sub-string with random values. In the second case, the last extra species are ignored. In this way, the algorithm can generate populations with different lengths. Figure 2 shows mutations in both parts of the population: *a*) mutation at the variable part: the first species of the variable part is mutated from the value 3 to 4, then one species is added into the string, and from the value 3 to 2, the fourth species is ignored; *b*) mutation at the fixed part: a species of the second part is mutated to other value in its domain.

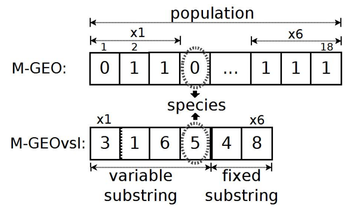

Figure 1: M-GEO × M-GEO$_{vsl}$: population

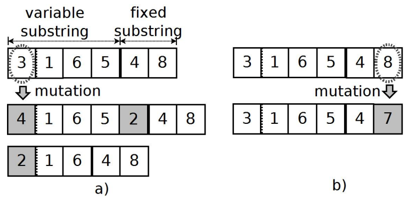

Figure 2: M-GEO$_{vsl}$: mutations

2.3 Evolution process

The evolution process of M-GEO$_{vsl}$ is illustrated in Figure 3. In each iteration of M-GEO$_{vsl}$, one objective function is randomly chosen with uniform distribution to be used as a function of fitness assignment. Thus, M-GEO$_{vsl}$ does not use directly the concept of non-dominance as criterion to guide the search used in Multi-objective Evolutionary Algorithms (MOEA) like NSGA-II, NPGA, MOGA and SPEA [6], but uses a rather simpler strategy similar to an early MOEA, VEGA [6], together with a re-initialization scheme in order to increase the possibility of finding solutions over the entire Pareto front.

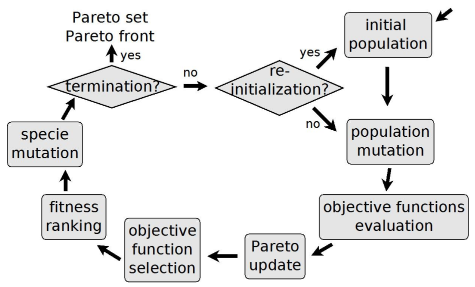

Figure 3: M-GEO$_{vsl}$: evolution process

The practical implementation of M-GEO$_{vsl}$ is as follows.

1. Initialize the population: for the first part of the population, generate a random integer value R for the first species and initialize randomly a string of length R; and for the second part, if exists, initialize randomly a string of length S. Then the population $\vec{x}$ consist of $R+1$ species of the first part, plus S species of the second part: $\vec{x} = x_0, x_1, \ldots, x_R, x_{R+1}, \ldots, x_{R+S}$.
2. Mutate each species i to other value mut_i of its domain, one at a time, generating $\vec{x_i}$. All objective functions evaluate $\vec{x_i}$, calculating $F(\vec{x_i}) = [F_1(\vec{x_i}), \ldots, F_{nf}(\vec{x_i})]$, where nf is the number of objective functions. After this evaluation, the value of the species i returns to its original one. This process is repeated for all species. Then all nf objective functions are evaluated to all $\vec{x_i}$, where $i = \{0, 1, \ldots, R+S\}$. Using $\vec{x_i}$, the approximation of the Pareto front is verified and updated, as necessary to contain only non-dominated solutions, and is kept in a separated archive. A solution $\vec{x_p}$ is said to dominate a solution $\vec{x_q}$ if and only if $F_i(\vec{x_p}) \leq F_i(\vec{x_q}), \forall i \in \{1, \ldots, nf\} \wedge F_j(\vec{x_p}) < F_j(\vec{x_q}), \exists j \in \{1, \ldots, nf\}$, for minimizing $F(\vec{x}) = [F_1(\vec{x}), \ldots, F_{nf}(\vec{x})]$.
3. Choose randomly one objective functions F_c, where $c \in \{1, \ldots, nf\}$ and nf is the number of objective functions.
4. Associate a fitness value to each species i as $\Delta_i = F_c(\vec{x_i}) - ref$, where ref is a given reference value. The fitness indicates the relative gain (or loss) in mutating the species, compared to a reference value (*e.g.*, zero).
5. Rank the species according to their fitness value. The first position ($k = 1$) of the ranking belongs to the least adapted species. For a minimization problem, lower values of Δ_i will have lower ranking. If two or more species have the same fitness, rank them randomly with uniform distribution.
6. Choose with uniform probability a candidate species i to mutate. Generate a random number RAN with uniform distribution in the range [0,1]. If $P_i(k) = k^{-\tau}$ is equal or greater than RAN, the species is confirmed to mutate. Tau (τ) is a free parameter set by the user and it regulates the determinism of the search. If $\tau = 0$, any variable has the same probability to be mutated. On the other hand, the higher values of τ, the more deterministic becomes the search to mutate the least adapted variable. In practice, it has been observed that the values used for τ which maximize the efficiency of the algorithm lie in the range [1,5]. A new candidate species is chosen until a species is confirmed to mutate. Set the population $\vec{x}$ to $\vec{x_i}$ obtained in the step 2.
7. Verify whether the stop criterion is reached. The stop condition is the maximum number of evaluations of all objective functions.
8. Verify whether a re-initialization of the population should be started. The condition to perform a new re-initialization is $numEval = int(maxNumEval/numInd)$, where $numEval$ is the current number of evaluations of all objectives functions, $maxNumEval$ the maximum number of objective function evaluations and $numInd$ the desired number of re-initializations. In the affirmative case, the algorithm returns to step 1 and a new population is randomly generated, keeping the solutions of the Pareto front. Otherwise, the algorithm returns to step 2.
9. Return the Pareto set and the Pareto front.

To illustrate the steps of the algorithm, consider the problem of minimizing the following objective functions in relation to an integer array $\vec{x}$ of length n and an integer y:

$$Minimize: F_1(\vec{x}, y) = \sum_{j=1}^{n} x_j^y \quad (1)$$

$$F_2(\vec{x}, y) = \sum_{j=1}^{n} (x_j - 2)^y \quad (2)$$

A simple numerical example is shown in Table 1 for this problem. In this case, the population represents the array $\vec{x}$ (variable substring) and the number y (fixed substring). Firstly, an integer array ($\vec{x}$) of length 4 and an integer value (y) are randomly generated, then in this moment the population has 6 species. In the second step, all species are mutated temporarily, one at a time, to rank the fitness of the species. Note that the first species, which represents the array length, is mutated from 4 to 6. Thus two integer values are randomly chosen to complete the array. It is worth noting that before the next species mutates, the early species returns to the original condition. Each new configuration is evaluated by both objective functions, obtaining a candidate point p for the Pareto front. If p is a non-dominated solution, then p is included in the approximation of the Pareto front and other solutions dominated by p are removed. In the example, the objective function F_2 is chosen in Step 3. In the next step, the fitness of the species are determined, calculating Δ_i with $ref = 0$. Then, all species are ranked by their fitness value in relation to the selected objective function F_2. Note that the ranking position of the first species is $k = 6$ and of the last one is $k = 1$. It indicates that the first species is better adapted than the last species, since mutating the first species, F_2 improves less than mutating the last one.

Table 1: Numerical example of M-GEO$_{vsl}$ steps

Step	Population	F_1	F_2	F_c	Δ_i	k
1	*Initialize the population*					
	4 $\mid$ 3 8 10 7 $\parallel$ 2	222	126	–	–	–
2	*Calculate F_1 and F_2 for each species mutation*					
	6 $\mid$ 3 8 10 7 9 1 $\parallel$ 2	304	176	–	–	–
	4 $\mid$ *1* 8 10 7 $\parallel$ 2	214	126	–	–	–
	4 $\mid$ 3 *3* 10 7 $\parallel$ 2	167	91	–	–	–
	4 $\mid$ 3 8 *2* 7 $\parallel$ 2	126	62	–	–	–
	4 $\mid$ 3 8 10 *5* $\parallel$ 2	198	110	–	–	–
	4 $\mid$ 3 8 10 5 $\parallel$ *1*	28	20	–	–	–
3	*Select an objective function*					
	4 $\mid$ 3 8 10 7	222	126	F_2	–	–
4	*Find the fitness of each species*					
	6 $\mid$ 3 8 10 7 9 1 $\parallel$ 2	304	176	F_2	176	–
	...					
	4 $\mid$ 3 8 10 5 $\parallel$ *1*	28	20	F_2	20	–
5	*Rank the species according to its fitness*					
	4 $\mid$ 3 8 10 5 $\parallel$ *1*	28	20	F_2	20	1
	...					
	6 $\mid$ 3 8 10 7 9 1 $\parallel$ 2	304	176	F_2	176	6
6	*Choose a candidate species to mutate*					
	4 $\mid$ 3 *3* 10 7 $\parallel$ 2	167	91	F_2	91	3
7	*Verify if termination criterion is reached*					
8	*Verify if it is necessary a re-initialization*					
9	*Return the Pareto set and Pareto front*					

In Step 6, one species is selected to be mutated according to the pressure of τ on the ranking. In the example, $\tau = 1.0$ is used and the third species with rank position $k = 3$ (not the least adapted) is chosen to mutate. The mutation is confirmed with $RAN = 0.2$ because the probability $P(3) = \frac{1}{3} = 0.33$ is greater than RAN. In this way, the population length remains with 6 species. However, if the first species was selected to mutate, the population would be increased to 8 species. When the maximum number of evaluations of all objective functions is reached, the Pareto set and Pareto front is returned. Otherwise, it is verified whether a re-initialization should start. Re-initialization is a common resource in multi-objective approach to avoid local optimal. Note that M-GEO$_{vsl}$ has only two free parameters that influence the solutions generation: τ and the number of re-initializations. In this way, it reduces effort to adjust the algorithm to a new problem.

3. M-GEO$_{VSL}$ APPLICATION

In this section, we present the application of M-GEO$_{vsl}$ in the context of test case generation. In the following, an approach to generate test case from EFSM is proposed using M-GEO$_{vsl}$. The EFSM definition and the experiments to evaluate the approach are also described.

3.1 The model

The system behavior is represented by an EFSM, defined as a tuple (S, s_0, I, O, T, V, P) in which [2]: S is a finite set of states; $s_0 \in S$ is the initial state; I is a set of input events; O is a set of output events; T is a finite set of transitions; V is a set of variables; and P is a set of input parameters. Each transition $t \in T$ is given by a source state $source(t) \in S$, a target state $target(t) \in S$ and a label of the form $i(t)[g(t)]/a(t)$ where: $i(t) \in I$, $g(t)$ is a logical expression called guard and $a(t)$ is the action executed when the transition is activated. Input events can contain one or more parameters belonging to P. The parts g and a of the label are optional. The model can be in only one of the states at moment. To change the state, it is necessary to trigger a transition. For a transition to be triggered, the corresponding input event should be received in the current state and the associated guard should be satisfied. Then the corresponding action a is executed, which may contain assignment statements or produce output events. When an **unexpected input** (i.e., input not specified in a given state) is received, the machine remains in its current state and generates a null output as response. A transition path is a sequence of transitions that, for every consecutive pair of transitions (t_j, t_{j+1}), $target(t_j) = source(t_{j+1})$. In order to illustrate the concepts presented, we use as example the EFSM M_1 of a vending machine [2] (Figure 4).

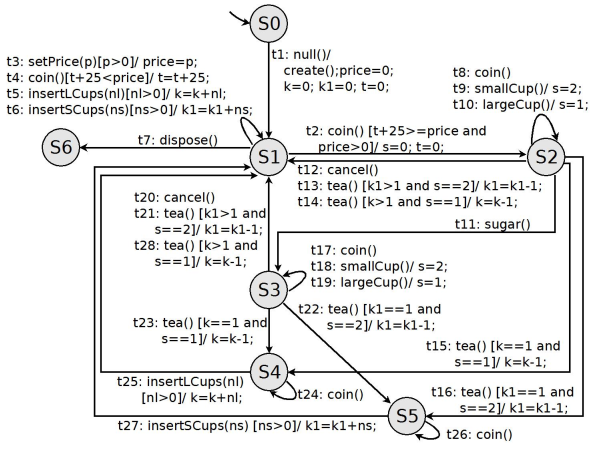
S0
t1: null()/ create();price=0; k=0; k1=0; t=0;
t3: setPrice(p)[p>0]/ price=p;
t4: coin()[t+25<price]/ t=t+25;
t5: insertLCups(nl)[nl>0]/ k=k+nl;
t6: insertSCups(ns)[ns>0]/ k1=k1+ns;
t8: coin()
t9: smallCup()/ s=2;
t10: largeCup()/ s=1;
S6
t7: dispose()
S1
t2: coin() [t+25>=price and price>0]/ s=0; t=0;
S2
t12: cancel()
t13: tea() [k1>1 and s==2]/ k1=k1-1;
t14: tea() [k>1 and s==1]/ k=k-1;
t20: cancel()
t21: tea() [k1>1 and s==2]/ k1=k1-1;
t28: tea() [k>1 and s==1]/ k=k-1;
t11: sugar()
t17: coin()
t18: smallCup()/ s=2;
t19: largeCup()/ s=1;
S3
t22: tea() [k1==1 and s==2]/ k1=k1-1;
t23: tea() [k==1 and s==1]/ k=k-1;
t15: tea() [k==1 and s==1]/ k=k-1;
S4
t25: insertLCups(nl) [nl>0]/ k=k+nl;
t24: coin()
t16: tea() [k1==1 and s==2]/ k1=k1-1;
S5
t27: insertSCups(ns) [ns>0]/ k1=k1+ns;
t26: coin()

Figure 4: EFSM M_1 for a vending machine

3.2 The test case generation approach

For the problem of generating test case from EFSM, the population in M-GEO$_{vsl}$ represents a sequence of input events and their parameters. As explained in Section 2.1, the population has a variable size part and a fixed size part. The variable part is formed by the sequence length and the sequence of input events, and the fixed size contains the parameters of all input events of the model. Then the population length is variable in relation to the input sequence length. Each parameter involved in the EFSM under consideration has a corresponding species in the population. We assume that the input parameters with identical name are the same, in this way the number of parameters is constant. Hence, when the parameters in different input events should not have the same values, different names should be given to them. It is the case of the input events $insertLCups(nl)$ and $insertSCups(ns)$ of M_1 (Figure 4) that add cups to the vending machine, but each of them has the respective parameter. Considering each input event encoded by an integer number, a population for M_1 can be:

|seq|	input event sequence	parameters: nl, ns, p
9	7 0 4 2 7 1 7 2 9	25 93 14

that represents an input sequence with 9 input events ($seq_1 = \{smallCup()^*, null(), insertSCups(93), setPrice(14), smallCup()^*, coin(), smallCup(), setPrice(14)^*, tea()\}$) and all 3 parameters of M_1.

The test cases derived from EFSM correspond to transition paths triggered by input sequences. Instead of the static approaches used by traditional techniques to generate the test cases, we use an executable model to produce the transition path dynamically during the execution of the EFSM. The executable model implements the behavior of an EFSM in a programming language and has some advantages: *i.* the user can validate the model before start generating test cases; *ii.* executing the model during the evolution process is more cost effective than executing the actual implementation, since the model abstracts away many details, such as access to a database or communication through a network. *iii.* the test case can be generated even when the source code is not available, like third-party components.

The interaction between the executable model and M-GEO$_{vsl}$ is illustrated in Figure 5. M-GEO$_{vsl}$ generates an input sequence and the executable model informs the path triggered by this sequence, taking into account all the data involved in the guard conditions. Then, M-GEO$_{vsl}$ evaluates the traversed path according to the test criterion. To guide the algorithm, dependence analysis of the EFSM is used to direct the search toward the test criterion.

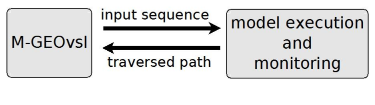

Figure 5: Interaction between M-GEO$_{vsl}$ and the executable model

The executable model is instrumented to produce the transition path triggered by the generated input sequence. When a transition t is executed, the input event $i(t)$ is received and the guard $g(t)$ is verified, considering the current state s and $source(t) = s$. The action $a(t)$ is executed, if $g(t)$ is true; otherwise, a null output is produced. In case an unexpected event is received, the model is executed according to the completeness assumption. Since the data to trigger a transition path is generated and considered during model execution, we prevent the generation of infeasible paths, at least at the model level.

It is worth noting that the terms input sequence and test case are distinct. An **input sequence** consists of a sequence of input events and their parameters. A **test case**, on the other hand, is the transition path triggered during model execution according to the given input sequence. Since the machine can be incomplete, i.e., the input sequence can contain unexpected events, the path length is not necessarily equal to the sequence size. In the example above, the unexpected inputs in seq_1 are marked with the symbol "*". The path triggered by seq_1 ($path = \{t_1 t_6 t_3 t_2 t_9 t_{13}\}$) has 6 transitions while the sequence length is 9.

In summary, the steps of the test case generation approach using M-GEO$_{vsl}$ are the following: 1. elaborate an EFSM M to represent the behavior of the system under test, based on its specification; 2. obtain an executable model of M; 3. validate M; 4. analyze the dependences of M; 5. generate the test cases using M-GEO$_{vsl}$.

To guide the test case generation, two objective functions are used in M-GEO$_{vsl}$: the test purpose coverage (F_1) and the minimum length of the input sequence (F_2). Due to the cost of test case execution time, the trade-off is to find a minimum length of the input sequence but long enough to cover the test purpose. We consider the test purpose as a target transition of the model to be covered. These objective functions are described in a previous work [22]; here we give an overview, for the sake of completeness. F_2 intends to minimize the input sequence length. A value into the range [0,1] is added to the sequence size in the sense of penalizing unexpected inputs, since no transitions are taken with these inputs.

$$\begin{aligned} Minimize: F_1 &= AL + ND & (3)\\ F_2 &= |seq| + (1 - 1.001^{-unexpected\_inputs}) & (4)\end{aligned}$$

where

$$\begin{aligned} AL &= 2 * |T_{affecting}| - RW \\ ND &= 1 - 1.001^{-d} \\ RW &= \begin{cases} |T'_{affecting}| & \text{if not cover } t; \\ |T'_{affecting}| + |T_{affecting}| & \text{if cover } t. \end{cases}\end{aligned}$$

F_1 is the function that represents information about the model in order to guide the search toward the test purpose. For F_1 evaluation, we define the sets of transitions $T_{affecting}$ and $T_{critical}$ using the information obtained from the dependence analysis of the model, which identifies the control and/or data relationships among the transitions. It is important to emphasize that these transitions sets are obtained in an early step of the evolution process. Considering a transition t, $T_{affecting}(t)$ contains all transitions upon which t is directly or indirectly control and/or data dependent, based on the dependence graph. The idea is to identify the transitions that could potentially affect t.

DEF. 1. *$T_{affecting}(t)$ consists of the transition t and the transitions obtained by a backward traversal of the dependence graph starting at t.*

To determine $T_{critical}$, we extend the concept of critical branching node [17] for an EFSM transition. In code-based testing, this is a branching node with an exit that, if taken, the test path misses the target.

DEF. 2. *A transition t_c is* ***critical*** *with respect to a target transition t if $t_c \notin T_{affecting}(t)$ and $\exists t_a \in T_{affecting}(t)$ such that $source(t_c) = source(t_a)$ and there exists a path π such that $t_c \in \pi$ and $t \notin \pi$.*

An algorithm to find the critical transitions for a given test purpose t is shown in Figure 6. In order to penalize the solution that takes a critical transition, F_1 uses a penalty value d. To determine the value of d, the transitions $t_a \in T_{affecting}(t)$ with the same source state of a critical transition (t_c) need to be analyzed and two situations need to be considered: *i.* the input event of t_c and t_a are the same but the guards are different and *ii.* the input event of t_c and t_a are different. In the first case, the distance d is computed using the functions defined by Tracey et al. [19]. In the second case, taking t_c receives a penalty γ in order to distinguish solutions with different input events.

```
input: test purpose $t$
$T_a = getTaffecting(t)$
for $\forall t_i \in T_a$ do
    $S = getSiblings(t_i)$
    for $\forall t_j \in S$ do
        if $t_j \notin T_a$ then
            if $event(t_j) == event(t_i)$ then
                $addTcritical(t_j, distanceFunction(t_i))$
            else
                $addTcritical(t_j, \gamma)$
            end if
        end if
    end for
end for
output: $T_{critical}(t)$
```

Figure 6: Algorithm to determine $T_{critical}$

Using $T_{affecting}$ and $T_{critical}$, the terms of F_1 can be calculated: approach level AL and normalized distance ND. The approach level AL measures how close an input sequence is to reach a path that traverses the test purpose t. The value of AL is related to the number of transitions of $T_{affecting}$ that were triggered ($|T'_{affecting}|$) during model execution. The triggered transitions of $T_{affecting}$ are used to minimize AL. If the sequence produces a path that traverses t (i.e., t is covered), the fitness value is rewarded with $|T_{affecting}|$. Then, the transitions $t_a \in T_{affecting}(t)$ are used to guide the search toward t. The normalizing distance ND is calculated at the point where the control flow takes a critical transition $t_c \in T_{critical}(t)$, that diverges away from a transition $t_a \in T_{affecting}(t)$. The term d in ND is defined as described earlier. The concepts of AL and ND are inspired from the SBST for code-based testing [16]. In general, AL is calculated verifying if a given path is traversed and, in contrast, we use the transitions of $T_{affecting}$ to guide the search. The term ND is based on the work of Baresel et al. [3]. However, to determine d, we need to consider that more than two transitions can have the same source state in an EFSM, in contrast to code-based testing that only the true and false branch of a node are verified.

3.3 Experiments

The experiments concern the following research questions:

Q_1**:** How to guide the decision maker to select the test cases?

Q_2**:** How well do the generated test cases result in comparison to existing approaches?

Table 2 describes the subject models in terms of number of states ($|S|$), transitions ($|T|$), input events ($|I|$) and parameters ($|P|$), and also their CCS (Cyclomatic Complexity of State machine) [20]. CCS is an adaption of metrics for design complexity of state models: $CCS = |T|+|I|+|A_G|+2$, where A_G is the set of atomic expressions in the guards. The models represent different sources. The first three models were used in [2] and the last one in [13].

Table 2: Experimental Models

Models	|S|	|T|	|I|	|P|	CCS
M_1 : vending machine	7	28	11	3	65
M_2 : cashier	12	21	15	7	51
M_3 : fuel pump	13	25	16	5	63
M_4 : transport protocol	6	21	13	14	57

3.3.1 Experimental set up

Before starting the test case generation, M-GEO$_{vsl}$ was adjusted for the problem being tackled. To tune the control parameter τ, the range of [1,5] was used with increment of 0.25. The stopping condition was defined as 10^5 function evaluations with 50 re-initializations. For F_1, the term d was calculated with $K = 100$ and $\gamma = 1000$. In relation to the population domain, the first element that represents the input sequence length is a positive integer smaller than 100. The input event domain is related to the input alphabet I of each model, being values from 1 to $|I|$. The parameter values are defined as positive integers. The number of parameters in the population is presented in Table 2. We used a Pentium 4 with 3.00 GHz and 1 GB of RAM memory.

The influence of the parameter τ in M-GEO$_{vsl}$ can be seen in Figure 7. It indicates that an intermediate value of τ in the range [1, 5] gives better results, meaning that a too deterministic ($\tau = 5.0$) or too random ($\tau = 1.0$) search is less prone to find the Pareto front for this problem.

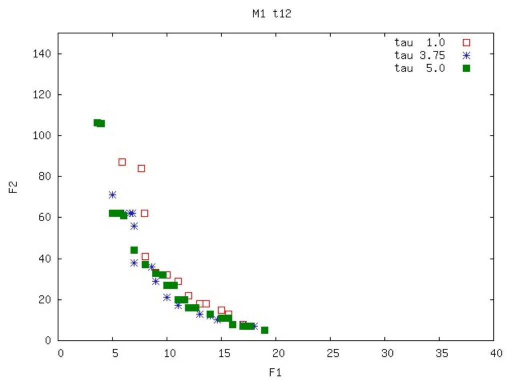

Figure 7: Influence of τ in M-GEO$_{vsl}$

Once obtained $\tau = 3.75$ as the best result in the tuning process, M-GEO$_{vsl}$ was executed to cover each transition of the subject models. We performed 10 runs per test purpose with 10^6 objective function evaluations and 100 re-initializations. The input sequence could be composed of 500 input events at maximum. It is important to clarify that this maximum value is used to avoid the generation of too long sequences. Considering the subjects, it is already a high value for the sequence length. The domain of the other population elements and the constants of the term d were defined as before. To evaluate the results for each transition, we build only one Pareto front from all fronts obtained in each run, plotting the non-dominated points in relation to all solutions generated in all runs.

For the testing approach, two tools were used: SMC[1] (State Machine Compiler) tool to obtain the executable models, and SLIM (SLIcing state based Model) tool [2] for the control and data dependence information. The source code of the executable model is in Java, in order to keep the language compatibility used in the test generator prototype.

3.3.2 Results

In relation to the cost of the test case generation approach, Table 3 shows the cost for all transitions of each model: the transition coverage, the average number of points in the Pareto front (Avg. Size) and its standard deviation (σ_p), and the average number of evaluations of the objective functions to find the solution (Avg. Eval.) and its standard deviation (σ_e). For all models, we obtained 100% of transition coverage, which means that the algorithm was able to find solutions to each transition of the model. Then, a Pareto front was produced with at least one point, for each target transition. Each point represents a successful path that covers the test purpose. It is worth noting that although the stopping condition for M-GEO$_{vsl}$ was 10^6 evaluations, less than half of this number was necessary to obtain a solution, on average. Each run of M-GEO$_{vsl}$ took approximately 22.28 seconds.

Table 3: Average results for all models

Model	Coverage	Avg. Size	σ_p	Avg. Eval.	σ_e
M_1	100%	16.21	4.98	510913.61	61999.98
M_2	100%	7.29	3.68	423318.67	183100.70
M_3	100%	2.84	1.65	486978.85	247153.34
M_4	100%	15.33	3.73	496579.78	104922.17

Q_1**:** To guide the tester in her/his decision, the Pareto front and the Pareto set for each transition can help. For instance, Figure 8 shows the Pareto front for the transition t_8 of M_1. The set of solutions are the trade-off between the objective functions F_1 and F_2. Table 4 provides information of some points of the front: the generated path, the path length (|Path|), the corresponding input sequence length (|Seq|) and the number of objective function evaluations to find the solution (#Eval). In case of minimizing F_2, the shortest input sequence can be obtained whereas, probably, few transitions of $T_{affecting}$ are covered. On the other hand, more transitions of $T_{affecting}$ can be taken minimizing F_1 but the input sequence can have a long length. As more than one solution can be generated for each test purpose, the test team plays the role of the decision maker to select the solution. The more solutions in the Pareto front, the more alternatives the test team have. Shortest test cases can be selected, when test case execution time is high. For this intent, the shortest path to t_8 (path p_1 of Table 4) can be used. When test case execution time is not an issue, the

[1]Available in http://smc.sourceforge.net.

tester can opt for the longest test cases. Since they contain more steps (input interactions), more transitions can be verified at once. For instance, it is possible to traverse 15 different transitions using the path p_5.

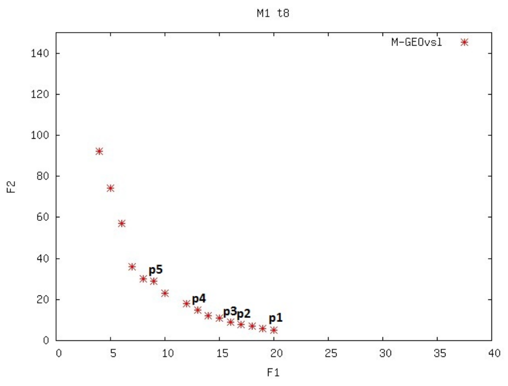

Figure 8: M_1: Pareto front for t_8

Table 4: Pareto set for t_x of M_1

Path		|Path|	|Seq|	#Eval
p_1 :	$t_1t_3t_2t_8$	4	5	599550
p_2 :	$t_1t_5t_3t_5t_2t_9t_8t_{10}$	8	8	866520
p_3 :	$t_1t_3t_5t_6t_2t_8t_{10}t_{14}$	8	9	885326
p_4 :	$t_1t_5t_3t_6t_4t_3t_2t_9t_8t_{10}t_{10}t_9t_{11}$ $t_{19}t_{19}$	15	15	247502
p_5 :	$t_1t_6t_6t_6t_6t_6t_3t_6t_2t_8t_{12}t_6t_5t_2t_9t_8$ $t_{13}t_2t_{10}t_{11}t_{28}t_6t_2t_{11}t_{19}t_{18}t_{20}t_3$	28	29	995738

Q_2**:** The work of Kalaji et al. [13, 12] is one of the closest SBST approach based on the EFSM model we could find of our approach for comparison purposes. All paths found in [13] to cover each transition of the model M_4 were used to compare with our approach. The main differences of their approach with respect to ours are: *i)* a genetic algorithm was used with a single objective optimization. They performed 1000 generations with a population of 25 individuals. *ii)* the test case length was not part of the evolution process in their approach. It was fixed in 11 for the model M_4; *iii)* the test cases generation occurred in two steps. In the first, they generated the transition paths, and in the second they randomly generated the test data to trigger those paths. Each path found in [13] was evaluated using F_1 and F_2, obtaining the point p_k in the objective function space. We analyzed p_k in relation to the Pareto front produced by M-GEO$_{vsl}$. To illustrate, Figure 9 shows the Pareto front obtained by M-GEO$_{vsl}$ for t_5 and the point p_k (indicated by ■), which is a dominated solution, hence not being an optimal solution for the problem. On the other hand, for transition t_6, the point p_k is a non-dominated solution in the objective function space, as presented in Figure 10. In fact, it dominates 4 solutions on the Pareto front found by M-GEO$_{vsl}$. For each transition of M_4, Table 5 shows the number of solutions on the Pareto front found by M-GEO$_{vsl}$ (pf), and the percentage of solutions (nd) on this front not dominated by the solution p_k, for each transition. For 9 of the 20 transitions to be covered, the Pareto front obtained by M-GEO$_{vsl}$ dominated the solutions found by Kalaji et al.. For the other transitions, the points p_k are non-dominated solutions, hence they can be considered optimal solutions from a multi-objective perspective.

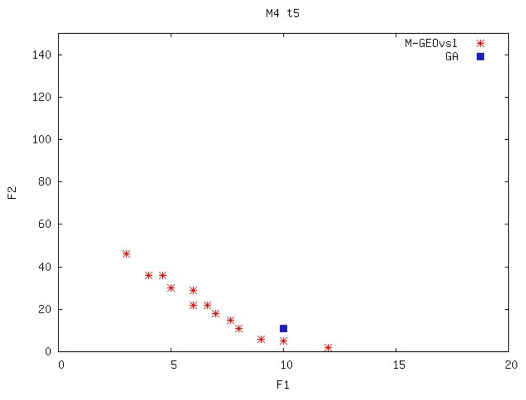

Figure 9: M_4: Pareto front for t_5

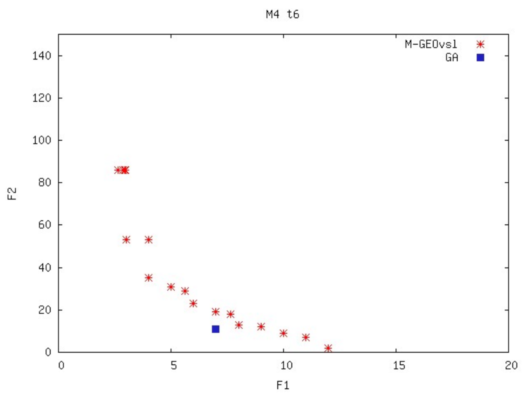

Figure 10: M_4: Pareto front for t_6

Table 5: M_4: Number of solutions on the Pareto front found by M-GEO$_{vsl}$ (pf) and percentage of solutions that are not dominated by p_k (nd)

Trans.	pf	nd (%)	Trans.	pf	nd (%)
t_0	20	95.00	t_{11}	5	100.0
t_1	19	89.47	t_{12}	15	100.0
t_2	14	100.0	t_{13}	14	92.86
t_3	19	79.94	t_{14}	12	91.67
t_4	13	84.61	t_{15}	17	94.12
t_5	13	100.0	t_{16}	19	100.0
t_6	18	77.78	t_{17}	13	92.31
t_7	15	80.00	t_{18}	20	100.0
t_8	14	64.28	t_{19}	11	100.0
t_9	14	92.85	t_{20}	17	100.0
t_{10}	20	100.0			

4. CONCLUSIONS

The steps of the algorithm of a new multi-objective implementation of the GEO method (M-GEO$_{vsl}$) were presented. M-GEO$_{vsl}$ allows the variation of the size of the string that encodes the design variables. It must be pointed out that although M-GEO$_{vsl}$ was developed in a specific context, it can in principle be applied to any multi-objective problem where the number of design variables is itself a variable of the problem.

An approach to generate test cases from EFSM using M-GEO$_{vsl}$ was described, presenting the following contributions: *i.* a dynamic approach is used for model-based test case generation, in which the model is the artifact that is executed, instead of the implementation of system under test. Then, test cases can be generated early in the development process, or in any situation in which source code is not available; *ii.* the transition paths, together with the data that trigger them, are obtained during evolution process. This intends to avoid infeasible path generation, at least at the model level; *iii.* a multi-objective approach is proposed not only to cover the test purpose, but also to minimize the test case length. In this way, it is no longer up to the customer to decide which test case length is better for a given target transition, as M-GEO$_{vsl}$ can determined the length automatically; *iv.* dependence analysis is used to guide the search for solutions.

In the experiments, a hundred percent of test purpose coverage was obtained for all benchmark models and also most of the generated solutions presented better results than a related work. A further work is to consider only the parameters involved in the generated input sequence, instead of optimizing all parameters of the model.

5. ACKNOWLEDGMENTS

This research is supported by CAPES, CNPq and Serasa Experian. The authors also wish to thank Zheng Li for providing the dependence analysis of the models.

6. REFERENCES

[1] B. T. Abreu, E. Martins, and F. L. Sousa. Generalized extermal optimization: An attractive alternative for test data generation. In *Proc. GECCO '07*, page 1138, 2007.

[2] K. Androutsopoulos, N. Gold, M. Harman, Z. Li, and L. Tratt. A theoretical and empirical study of EFSM dependence. In *Proc. ICSM'09: 25th IEEE Int. Conf. on Software Maintenance*, pages 287–296, Sept. 2009.

[3] A. Baresel, H. Sthamer, and M. Schmidt. Fitness function design to improve evolutionary structural testing. In *Proc. GECCO'02*, pages 1329–1336, 2002.

[4] S. Boettcher and A. G. Percus. Optimization with extremal dynamics. *Physical Review Letters*, 86:5211–5214, 2001.

[5] A. V. Buzzo, E. Martins, and F. L. De Sousa. Using georeal algorithm for automated test data generation. In *SAST'2010: 4th Brazilian Workshop on Systematic and Automated Software Testing*, pages 109–124, 2010. (in Portuguese).

[6] C. A. C. Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1:28–36, Feb. 2006.

[7] F. L. De Sousa, F. M. Ramos, P. Paglione, and R. M. Girardi. New stochastic algorithm for design optimization. *AIAA Journal*, 41(9):1808–1818, Sept. 2003.

[8] F. L. De Sousa, V. Vlassov, and F. M. Ramos. Generalized extremal optimization for solving complex optimal design problems. In *Proc. GECCO'03*, volume 2723 of *Lecture Notes in Computer Science*, pages 375–376, 2003.

[9] F. L. De Sousa, V. Vlassov, and F. M. Ramos. Generalized extremal optimization: An application in heat pipe design. *Applied Mathematical Modelling*, 28(10):911–931, Oct. 2004.

[10] R. L. Galski. *Development of improved, hybrid, parallel and multiobjective versions of the generalized extremal optimization method and its application to the design of spatial systems.* PhD thesis, INPE, São José dos Campos, SP, Brazil, 2006. (in Portuguese).

[11] M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, Department of Computer Science, King's College London, April 2009.

[12] A. S. Kalaji, R. Hierons, and S. Swift. Generating feasible transition paths for from an extended finite state machine (EFSM) with the counter problem. In *Proc. SBST'10: 3rd Int. Workshop on Search-Based Software Testing*, pages 232–235, 2010.

[13] A. S. Kalaji, R. M. Hierons, and S. Swift. Generating feasible transition paths for testing from an extended finite state machine. In *Proc. ICST'09*, pages 230–239, 2009.

[14] K. Lakhotia, M. Harman, and P. McMinn. A multi-objective approach to search-based test data generation. In *Proc. GECCO'07*, pages 1098–1105, 2007.

[15] R. Lefticaru and F. Ipate. Functional search-based testing from state machines. In *Proc. ICST '08*, pages 525–528, 2008.

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

[17] P. McMinn, M. Harman, D. Binkley, and P. Tonella. The species per path approach to search-based test data generation. In *Proc. ISSTA'06*, pages 13–24, 2006.

[18] P. McMinn and M. Holcombe. Evolutionary testing of state-based programs. In *Proc. GECCO'05*, pages 1013–1020, 2005.

[19] N. Tracey, J. Clark, J. McDermid, and K. Mander. A search-based automated test-data generation framework for safety-critical systems. *Systems engineering for business process change: new directions*, pages 174–213, 2002.

[20] S. Wagner and J. Jürjens. Model-based identification of fault-prone components. In *Proc. EDCC-5: Fifth European Dependable Computing Conference, volume 3463 of LNCS*, pages 435–452. Springer, 2005.

[21] T. Yano, E. Martins, and F. L. De Sousa. Generating feasible test paths from an executable model using a multi-objective approach. In *Proc. SBST'10: 3rd Int. Workshop on Search-Based Software Testing*, pages 236–239, 2010.

[22] T. Yano, E. Martins, and F. L. De Sousa. Most: a multi-objective search-based testing from efsm. In *Proc. SBST'11: 4th Int. Workshop on Search-Based Software Testing*, 2011.

[23] S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In *Proc. ISSTA '07*, pages 140–150, 2007.