Chapter 1

PRIORITIZING COVERAGE-ORIENTED TESTING PROCESS — AN ADAPTIVE-LEARNING-BASED APPROACH AND CASE STUDY

FEVZI BELLi
Department of Computer Science, Electrical Engineering and Mathematics
Institute for Electrical Engineering and Information Technology
University of Paderborn
Warburger Straße 100 D-33098 Paderborn, Germany
belli@upb.de

MUBARIZ EMINOV
Halic University, Faculty of Engineering
Department of Computer Engineering, Istanbul, Turkey
mubarizemunli@halic.edu.tr

NIDA GÖKÇE
Department of Statistics, Faculty of Science
Mugla University, 4800 Mugla, Turkey
gnida@mu.edu.tr

W. ERIC WONG
Department of Computer Science, University of Texas at Dallas
Richardson, Texas 75080, USA
ewong@utdallas.edu

This chapter proposes a graph-model-based approach to prioritizing the test process. Tests are ranked according to their preference degrees which are determined indirectly, i.e., through classifying the events. For construction of the groups of events, an unsupervised neural network is trained by adaptive competitive learning algorithm. A case study demonstrates and validates the approach.

1. Introduction and Related Work

Testing is one of the important, traditional analytical techniques of quality assurance in the software industry. There is no justification, however, for any assessment of the correctness of system under test (SUT) based on the success of a single test, because potentially there can be an infinite number of test cases. To overcome this principal shortcoming of testing concerning
For a productive generation of tests, model-based techniques focus on particular, relevant aspects of the requirements of the SUT and its environment. Real-life SUTs have, however, numerous features that are to simultaneously be considered, often leading to a large number of tests. In many applications where testing is required, the complete set of tests is not run due to time or budget constraints. In such cases, the entire set of system features cannot be considered. In these situations, it is essential to prioritize the test process. It is then essential to model the relevant features of SUT. The modeled features are either functional behavior or structural issues of the SUT (as given in its code), leading to specification-oriented testing or implementation-oriented testing, respectively. Once the model is established, it “guides” the test process to generate and select test cases, which form sets of test cases (also called test suites). The test selection is ruled by an adequacy criterion, which provides a measure of how effective a given set of test cases is in terms of its potential to reveal faults. Some of the existing adequacy criteria are coverage-oriented. They use the ratio of the portion of the specification or code that is covered by the given test set in relation to the uncovered portion in order to determine the point in time at which to stop testing (test termination problem).

Test case prioritization techniques organize the test cases in a test suite by ordering such that the most beneficial are executed first thus allowing for an increase in the effectiveness of testing. One of the performance goals, i.e., the fault detection rate, is a measure of how quickly faults are detected during the testing process.

This chapter is on model-based, specification- and coverage-oriented testing. The underlying model graphically represents the system behavior interacting with the user’s actions. In this context, event sequence graphs (ESG) are favored. ESG approach view the system’s behavior and user’s actions as events, more precisely, as desirable events, if they are in accordance with the user expectations, otherwise they are undesirable events. Mathematically speaking, a complementary view of the behavioral model is generated from the model given. Thus, the model will be exploited twice, i.e., once to validate the system behavior under regular conditions and a second time to test its robustness under irregular, unexpected conditions.

The costs of testing often tend to run out the limits of the test budget. In those cases, the tester may request a complete test suite and attempt to run as many tests as affordable, without running out the budget. Therefore,
it is important to test the most important items first. This leads to the
Test Case Prioritization Problem (TCPP) a formal definition of which is
represented in as follows:

Given: A test suite $T$; the set $PT$ of permutations of $T$; a function $f$ from
$PT$ to the real numbers which represents the preference of the
tester while testing.
Problem: Find $T' \in PT$ such that ($\forall T''$) ($T'' \neq T'$) [$f(T') \geq f(T'')$]

Existing approaches to solving TCPP usually suggest constructing a
density covering array in which all pair-wise interactions are covered. $^4,5$
Generally speaking, every $n$-tuple is then qualified by a number $n \in \mathbb{N}$ ($\mathbb{N}$: set of natural numbers) of values to each of which a degree of importance is
assigned. In order to capture significant interactions among pairs of choices
the importance of pairs is defined as the “benefit” of the tests. Every pair
covered by the test contributes to the total benefit of a test suite by its
individual benefit. Therefore, the tests given by a test suite are to be
ordered according to the importance of corresponding pairs. However, such
interaction-based, prioritized algorithms are computationally complex and
thus usually less effective. $^6,7$

The ESG approach favored in this chapter generates test suites through
a finite sequence of discrete events. The underlying optimization problem is
a generalization of the Chinese Postman Problem (CPP) $^8$ and algorithms
given in $^9$–$^{11}$ differ from the well-known ones in that they satisfy not only
the constraint that a minimum total length of test sequences is required, but
also fulfill the coverage criterion with respect to converging of all event pairs
represented graphically. This is substantial to solve the test termination
problem and accounts for a significant difference of this present chapter from
existing approaches. To overcome the problem that an exhaustive testing
might be infeasible, the present chapter develops a prioritized version of
the mentioned test generation and optimization algorithms, in the sense
of “divide and conquer” principle. This is the primary objective and the
kernel of this chapter which is novel and thus, to our knowledge, has not
yet been worked out in previous work.

The required prioritization has to meet the needs and preferences of
test management on how to spend the test budget. However, SUT and
software objects, i.e., components, architecture, etc., usually have a great
variety of features. Therefore, test prioritization entails the determination of
order relation(s) for these features. Generally speaking, we have $n$ objects,
whereby each object has a number ($p$) of features that we call dimension.
TCPP then represents the comparison of test objects of different, multiple dimensions. To our knowledge, none of the existing approaches take the fact into account that SUT usually has a set of attributes and not a single one when prioritizing the test process. Being of enormous practical relevance, this is a tough, \textit{NP}-complete problem.

Our approach assigns to each of the tests generated a degree of its preference. This degree is indirectly determined through estimation of the events qualified by several attributes. We suggest representing those events as an unstructured multidimensional data set and dividing them into groups which correspond to their importance. Beforehand, the optimal number of those groups is determined by using \textit{V}$_{sv}$ \textit{index-based clustering validity algorithm},\textsuperscript{12,13} To derive the groups of events we use a clustering approach based on unsupervised neural networks (NN) that will be trained by an adaptive competitive learning (CL) algorithm.\textsuperscript{14} Different from the existing approaches, e.g., as described in,\textsuperscript{12,15,16} input and weight vectors are normalized, i.e., they have length one. This enables less sensitivity to initialization and a good classification performance. The effectiveness of the proposed testing approach is demonstrated and validated by a case study a non-trivial commercial system.

The chapter is organized as follows. Section 2 explains the background of the approach, presenting also the definition of neural network-based clustering. Section 3 explains the CL algorithms. Section 4 describes the proposed prioritized graph-based testing approach. Section 5 includes the case study. Section 6 summarizes the results, gives hints to further research and concludes the chapter.

2. Background

2.1. Event Sequence Graphs

Because the construction of ESG, test generation from ESG and test process optimization are sufficiently explained in the literature,\textsuperscript{9–11,17} the present chapter summarizes ESG concept, as far as it is necessary and sufficient to understand the test prioritization approach represented in this chapter.

Basically, an \textit{event} is an externally observable phenomenon, such as an environmental or a user stimulus, or a system response, punctuating different stages of the system activity. A simple example of an ESG is given in Fig. 1. Mathematically, an ESG is a directed, labeled graph and may be thought of as an ordered pair \( \text{ESG} = (\alpha, E) \), where \( \alpha \) is a finite set of
nodes (vertices) uniquely labeled by some input symbols of the alphabet \( \Sigma \), denoting events, and \( E: \alpha \rightarrow \alpha \), a precedence relation, possibly empty, on \( \alpha \). The elements of \( E \) represent directed arcs (edges) between the nodes in \( \alpha \). Given two nodes \( a \) and \( b \) in \( \alpha \), a directed arc \( ab \) from \( a \) to \( b \) signifies that event \( b \) can follow event \( a \), defining an event pair (EP) \( ab \) (Fig. 1).

The remaining pairs given by the alphabet \( \Sigma \), but not in the ESG, form the set of faulty event pairs (FEP), e.g., \( ba \). As a convention, a dedicated, start vertex, e.g., \([\cdot] \), is the entry of the ESG whereas a final vertex e.g., \([\cdot] \) represents the exit. Note that \([\cdot] \) and \([\cdot] \) are not included in \( \Sigma \); therefore, the arcs from and to them form neither EP nor FEP. The set of FEPs constitutes the complement of the given ESG (\( \overline{ESG} \)). Superposition of ESG and \( \overline{ESG} \) leads to completed ESG (\( \hat{ESG} \)) (Fig. 1).

A sequence of \( n + 1 \) consecutive events that represents the sequence of \( n \) arcs is called a event sequence (ES) of the length \( n + 1 \), e.g., an EP (event pair) is an ES of length 2. An ES is complete if it starts at the initial state of the ESG and ends at the final event; in this case it is called a complete ES (CES). Occasionally, we call CES also walks (or paths) through the ESG given. A faulty event sequence (FES) of the length \( n \) consists of \( n - 1 \) subsequent events that form an ES of length \( n - 2 \) plus a concluding, subsequent FEP. An FES is complete if it starts at the initial state of the ESG; in this case it is called faulty complete ES, abbreviated as FCES. A FCES must not necessarily end at the final event.

### 2.2. Neural Network-Based Clustering

Clustering is a technique to generate an optimal partition of a given, supposedly unstructured, data set into a predefined number of clusters (or groups). Homogeneity within the groups and heterogeneity between them can be settled by means of unsupervised neural network-based clustering algorithms.\(^{13,14}\) For clustering of an unstructured data set dealing especially with vector quantization, unsupervised learning based on clustering in a neural network framework is frequently used. Clustering has to obtain
partition data vector space

\[ X = \{x_1, \ldots, x_i, \ldots, x_n\} \subset \mathbb{R}^p, \]
\[ x_i = (x_{i1}, \ldots, x_{ij}, \ldots, x_{ip}) \in \mathbb{R}^p \]  

(1)

into \( c \) number clusters or subspaces \( S_k \) in the form of hyper spherical clouds of pattern vectors \( x_i = \{x_1, x_2, \ldots, x_p\} \in \mathbb{R}^p \). Each of these subspaces is represented by a cluster center (prototype) that corresponds to weight vector \( w = (w_1, w_2, \ldots, w_p) \in \mathbb{R}^p \). An input (pattern) vector \( x_i \) is described by the best-matching or “winning” weight vector \( w_k \) for which criterion-distortion error \( d(x_i, w_k) = ||x_i - w_k||^2 \) that is the squared error of Euclidean distance is minimum. The procedure divides the input space \( \mathbb{R}^p \) into partition subspace

\[ S_k = \{X \in \mathbb{R}^p \| ||x - w_k|| \leq ||x - w_j|| \forall j \neq k\} \quad k = 1, \ldots, c \in \mathbb{N} \]  

(2)
called Voronoi polygons or Voronoi polyhedra. To determine the optimal number \( c \) of groups, the \( V_{sv} \) index-based cluster validity algorithm\(^{13}\) has been used.

To provide training of NN, a number of learning algorithms are used. We deal with the family of competitive learning (CL) algorithms\(^{14}\) that are a type of self-organizing networking models. According to CL algorithms, “winning” weight vector (weights of connections between input and output nodes) or cluster center can be adjusted by applying “winner-takes-all” strategy in training phase of the NN under consideration. In clustering a data set which is to be partitioned into \( c \) number of clusters each of which contains a data subset \( S_k \) defined as follows:

\[ X = \bigcup_{k=1}^{c} S_k \quad \text{with} \quad S_k \cap S_j = 0 \quad \forall k \neq j \]  

(3)

An optimal partition \( x_i \in \mathbb{R}^p \) into subspaces \( S_k \), \( k = 1, 2, \ldots, c \), is obtained through an optimal choice of reference vectors \( w_k \) which minimize a cost function-distortion error represented as follows:

\[ E = \sum_{k=1}^{c} \int_{S_k} d(x, w_k)g(x)dx \]  

(4)

where \( g(x) \) is a probability density function. If probability distribution of data vectors \( g(x) \) is known in advance then gradient descent algorithm can
be applied on $E$ in order to minimize (4) which leads to well-known $k$-means clustering algorithm\textsuperscript{18} under fixed number of subspaces (clusters). Thus, optimal weight vectors $w_k$, $k = 1, \ldots, c$, can be precisely determined. However, in general, $g(x)$ is not given a priori, therefore a number of neural network clustering algorithms\textsuperscript{15,16,19,20} were suggested to evaluate this unknown density function. In the case, when the number of the clusters $c$ increases, density function $g(x)$ in each cluster becomes approximately uniform,\textsuperscript{27} therefore (4) can be rewritten as

$$E = \sum_{k=1}^{c} g(w_k) \int_{S_k} d(x, w_k) dx$$

(5)

Let $D_k$ be partition error in $k$-th subspaces $S_k$, then

$$E = \sum_{k=1}^{c} D_k$$

(6)

where $D_k = g(w_k) \int_{S_k} d(x, w_k) dx$. As the sequence of input vectors becomes stationary and ergodic, it is known that (5) is corresponding to (7) presented in the mean square error (MSE) as follows\textsuperscript{16,19}

$$E = \frac{1}{n} \sum_{k=1}^{c} D_k$$

(7)

where

$$D_k = \frac{1}{p} \left( \sum_{x \in S_k} d(x, w_k) \right)$$

(8)

$n$ is the total number of input vectors and $p$ is the dimension of input vector. Therefore, the optimal clustering results in obtaining partition subspaces $S_k$ and weight vectors $w_k$, $k = 1, 2, \ldots, c$ that minimize $D_k$.

3. Competitive Learning

CL is a paradigm in which a structured or unstructured population of units compete with each other with regard to a stimulus, where the winner (or winners) of the competition may respond and be adapted. Algorithm that implements CL is suited to different specific concerns, although it is generally nonparametric, and suited to the general domains of function approximation, classification, and regression.\textsuperscript{21}
CL is a connectionist machine learning paradigm where an input pattern is matched to the node with the most similar input weights, and the weights are adjusted to better resemble the input pattern. This is called the *winner-take-all* (or maximum activation) unsupervised learning method where the input pattern is compared to all nodes based on similarity. The nodes compete for selection (or stimulation) and ultimately adjustment (or learning). Kohonen distinguishes this connectionist learning paradigm from feed-forward and feed-backward approaches as follows:

**Signal Transfer Networks**: (feed-forward paradigm) Signal transform circuits where the output signals depend on the input signals received by the network. Parametric in that the mapping is defined by a basis function (components of the structure) and fitted using an optimization approach like gradient decent. Examples include the multilayer Perceptron, back propagation, and radial basis function.

**State-Transfer Networks**: (feed-backward paradigm) Based on relaxation effects where the feedbacks and nonlinearities cause the activity state to quickly converge to one of its stable values (attractor). Input signals provide the initial activity state, and the final state is a result of recurrent feedbacks and computation. Examples include Hopfield network, Boltzmann machine, and bidirectional associative memory (BAM).

**Competitive Learning**: (self-organizing network paradigm) Networks of cells in simple structures receive identical inputs from which they compete for activation through positive and negative lateral interactions. One cell is the winner, and other cells are inhibited or suppressed. Cells become sensitive to different inputs and act as decoders for the domain. The result is a globally ordered map created via a self-organizing process. Examples include the Self-Organizing Map (SOM), and Learning Vector Quantization (LVQ).

Fritzke uses taxonomy of hard (*winner-take-all*) and soft (*winner-take-most*) CL and further distinguishes soft approaches to those with and without a fixed network topology.

**Hard CL**: Winner-take-all (WTA) learning each input signal results in the adaptation of a single unit of the model. These methods may occur online or offline in batch. Examples include k-means.

**Soft CL**: Winner-take-most (WTM) learning where an input signal results in the adaptation of more than one unit of the model. No fixed model dimensionality or topology is prescribed with these methods. Examples include neural gas.
Prioritizing Coverage-Oriented Testing Process

**Soft CL with Fixed Structure**: Winner-take-most (WTM) learning with a fixed model dimensionality and or topology. Examples include the self-organizing map.

### 3.1. Distance-Based Competitive Learning Algorithm

CL is closely related to clustering that learns to group input patterns in clusters. In the input space $x \in R^p$ the input pattern $x_i$ is defined, which is generated upon the probability density function $g(x)$, by applying the winner-take-all strategy. When both input vectors and weight vectors are not normalized, the Euclidean distance measure, in general, is used to determine the winner weight vector $w_w$ in CL algorithms.

**Training**: In the training phase the weight vectors of NN are updated usually according to Standard CL algorithm. Firstly, for a data point $x_i \in R^p$ selected from X the winner weight vector $w_w$ is determined by:

$$w_w = \arg \min_k \{\|x_i - w_k\|\}$$

$$i = 1, \ldots, n \in \mathbb{N} \quad k = 1, \ldots, c \in \mathbb{N}$$

(9)

where $\| . \|$ is *Euclidean distance measure*. Then this vector is adjusted at step $t$ by

$$\Delta w_w(t) = \eta(t)(x_i - w_w)$$

(10)

where $\eta(t)$ is a *learning rate*. Secondly, the adjusted winner vector is calculated by

$$w_w(t) = w_w(t - 1) + \Delta w_w(t)$$

(11)

Training process iteratively proceeds until the convergence condition for the all weight vectors is satisfied. Clearly, the CL algorithm actually seeks for a local minimum (with respect to the predetermined number of clusters) for squared error criterion by applying gradient descent optimization. As known, in the Kohonen’s SOFM algorithm not only $w_w$ but also the weight vectors that are placed in its neighborhood are adjusted. The learning rate for these weight vectors is set to be much smaller than the rate for $w_w$ that is slowly reduced up to the winner weight vector. Thus, the updating rule of this learning algorithm becomes as

$$\Delta w_k(t) = \eta(t)(x_i - w_k) \quad k \in N_c$$

(12)
where \( N_c(t) \) has a set of indexes of neighborhoods for the winner \( w_k \) at step \( t \). If \( N_c(t) \) has index of the winner only, then Kohonen’s algorithm becomes the standard CL algorithm presented in above.

### 3.2. Angle-Based Competitive Learning Algorithm

Now, we consider the updating rule of CL algorithm when both input vectors and the weight vectors are normalized to a unit length, that is, all vectors are presented as the unit vectors. For the input patterns \( x \in \mathbb{R}^p \), the corresponding normalized vectors \( \tilde{x} \) are given by

\[
\tilde{x} = \frac{x}{|x|} = x \left( \sum_{j=1}^{p} x_j^2 \right)^{-1/2}
\]  

(13)

where \( |x| \) is the magnitude of input vector \( x \) that lies on a unit hyper sphere in \( \mathbb{R}^p \). In this case, as known, the winner weight vector is determined by the dot product of the presented input vector \( x_i \) and a weight vector \( w_k \), then (9) can be reformed as the follows

\[
\tilde{w}_w = \arg \max_k \left\{ \sum_{j=1}^{p} \tilde{x}_{ij} \tilde{w}_{kj} \right\}
\]

(14)

i.e., the winner vector \( \tilde{w}_w \) is chosen by the largest activation level. Since the dot product is \( \cos \theta \) where \( \theta \) the angle between is two considered vectors, then (14) can be expressed as

\[
\tilde{w}^\theta_w = \arg \min_k \{ \theta_k \} \quad k = 1, \ldots, c
\]  

(15)

i.e., the winner vector \( \tilde{w}^\theta_w \) is determined by the smallest angle level between the presented \( x_i \) and weight vectors \( w_k \), \( k = 1, \ldots, c \). The updating rule of a winner weight vector instead of (10) is based on the adjusting equation (16) expressed as follows

\[
\Delta \tilde{w}_w(t) = \eta(t) \left( \frac{\tilde{x}_i}{p} - \tilde{w}_w \right)
\]  

(16)

Then for Kohonen’s SOFM algorithm the updating rule has the following form:

\[
\Delta \tilde{w}_k(t) = \eta(t) (\tilde{x}_i - \tilde{w}_k) \quad k \in N_c
\]  

(17)
Thus, the winner weight vector at step $t$ will be

$$\tilde{w}_w(t) = \tilde{w}(t - 1) + \Delta \tilde{w}_w(t)$$

(18)

However, in general, the distribution is not given in advance; hence the initial values of the weight vectors are randomly allotted. It negatively influences the clustering performance of the considered CL algorithm.

### 3.3. Adaptive Competitive Learning

In this section we present the CL algorithm for neural network clustering, which is able to: have limited dependence on initial values of weight vectors; reduce partition performance. Similar to the studies in\textsuperscript{15,16} disclosed above, it uses the deletion method that eliminates sequentially weight vectors which are prepared more than their predetermined numbers in advance. As in\textsuperscript{9} where this learning algorithm is called the adaptively CL, we use the simplest standard CL algorithm. However, instead of direct employing of the input vectors, it utilizes the corresponding vectors normalized to a unit length. Rummelhart\textsuperscript{26} introduced such kind of normalization of the vectors in input space for CL and afterwards it was used in the few versions of Kohonen’s SOFM algorithm.\textsuperscript{20} It has been utilized for classification, as well as shown good performance.\textsuperscript{27}

In the suggested CL algorithm we use a deletion method based on a criterion of subdistortion or intra-cluster partition error but in this case its equation will be different from (8) and it becomes as

$$D_k = \frac{1}{p} \left( \sum_{\bar{x} \in S_k} \bar{x} \tilde{w}_w \right)_{k = 1, \ldots, c}$$

(19)

The self-elimination procedure carried out according to (19) is shortly described as the follows. After learning by standard CL algorithm, a weight vector $w_i$ that has a minimum intra-cluster partition error, i.e., $D_k \geq D_k$, for all $k$, is deleted. Due to the use of the activation-based selection of the winner vector we call the suggested algorithm as the activation checking based CL with deletion. In this case, to signify subdistortion error as minimal we use the angle estimation of $D_k$ presented as the follows

$$D_k^* = \frac{1}{p} \left( \sum_{\bar{x} \in S_k} \theta_k \right)_{k = 1, \ldots, c}$$

(20)
Then using (20) and (7), clustering performance can be estimated by criteria such as the mean angle error and the standard angle deviation among subdistortions for given data set. Thus, the proposed Adaptive CL algorithm is presented as follows.

**Adaptive Competitive Learning Algorithm:**

**Step 1. Initialization:**
Initial number of output neurons $l_0$, final number of neurons $l$, maximum iteration $T_{\text{max}}$, initial iteration of deletion $t_0 = T_{\text{max}}/3$ and partial iteration $u = T_{\text{max}}/3(l_0 - l + 1)$, Set $t ← 0$ and $m ← l_0$

**Step 2. Angle-Based Competitive Learning:**
2.1. Choose an input vector $\tilde{x}_i$ at random among $X$
2.2. Select a winner $\tilde{w}_k$ according to (14)
2.3. Update the winner $\tilde{w}_w$ vector according to (16)
2.4. Set $t ← t + 1$
2.5. If $m > l$ and $t = t_0 + u \times q$ than go to Step 3, otherwise go to 2.1.

**Step 3. Deletion Mechanism:**
3.1. Delete $\tilde{w}_k$ calculating $D_k$ according to (19) and checking $D_s \geq D_k$
3.2. Set $m ← m - 1$

**Step 4. Termination Condition:**
If $t = T_{\text{max}}$ then terminate, otherwise go to Step 2.

**Classification:** After finding a value of weight vectors $\{w_1, \ldots, w_c\}$ that correspond to cluster centers, respectively, a data set is divided into $c$ groups as follows:

$$S_k = \left\{ x \in \mathbb{R}^p \mid \sum_{j=1}^{p} \tilde{x}_{ij}\tilde{w}_{kj} \geq \sum_{j=1}^{p} \tilde{x}_{ij}\tilde{w}_{mj} \forall k \neq m \right\} \quad (21)$$

$i = 1, \ldots, n$, $j = 1, \ldots, p$, $k = 1, \ldots, c$, $m = 1, \ldots, c \in \mathbb{N}$

Classification performance of the considered clustering algorithm was estimated by the MSE calculated using (7) and (19). Effectiveness of this algorithm was verified for different types of data sets in.\textsuperscript{14} Computational time for classification depends on the number $n$ of the events and the number $p$ of the attributes.
4. Prioritized ESG-Based Testing

We consider the testing process based on the generation of a test suite from ESG that is a discrete model of a SUT. To generate tests, firstly a set of ESGs are derived which are input to the generation algorithm to be applied. We deal with the test generation algorithms\(^9-11\) that generates tests for a given ESG and satisfies the following coverage criteria.

(a) Cover all event pairs in the ESG.
(b) Cover all faulty event pairs derived by the ESG.

Note that a test suite that satisfies the first criterion consists of CESs while a test suite that satisfies the second consists of FSESs. These algorithms are able to provide the following constraints:

(a) The sum of the lengths of the generated CESs should be minimal.
(b) The sum of the lengths of the generated FSESs should be minimal.

The constraints on total lengths of the tests generated enable a considerable reduction in the cost of the test execution and thus the algorithms mentioned above can be referred to as the relatively efficient ones. However, as stated in Section 1, an entire test suite generated may not be executed due to limited project budget. Such circumstances entail ordering all tests to be checked and exercised as far as they do not exceed the test budget. To solve the test prioritizing problem, several algorithms have been introduced.\(^1,4\) Usually, during the test process for each \(n\)-tuple (in particular pair-wise) interaction a degree of importance is computationally determined and assigned to the corresponding test case. However, this kind of prioritized testing is computationally complex and hence restricted to deal with short test cases only.

Our prioritized testing approach is based on the ESG-based testing algorithms mentioned above. Note that our test suite consists of CESs which start at the entry of the ESG and end of its exit, representing walks (paths) through the ESG under consideration. This assumption enables to order the generated tests, i.e., CESs.

The ordering of the CESs is in accordance with their preference degree which is defined indirectly, i.e., by estimation of events that are the nodes of ESG and represent objects (modules, components) of SUT. For this aim, firstly events are presented as a multidimensional event vector

\[ x_i = (x_1, \ldots, x_p) \]

where \(p\) is the number of attributes.
4.1. Definition of the Attributes of Events
To qualify an event corresponding to a node in ESG, as a arbitrarily chosen example, we propose to use following 9 attributes, i.e., \( p = 9 \), that determine the dimension of a data point represented in a data set.\(^{28,29}\) These attributes are given below:

\( x_1 \): The number of sub-windows to reach an event from the entry \([\text{gives its distance to the beginning}]\).

\( x_2 \): The number of incoming and outgoing edges (invokes usage density of a node, i.e., an event).

\( x_3 \): The number of nodes (events) which are directly and indirectly reachable from an event except entry and exit (indicates its “traffic” significance).

\( x_4 \): The maximum number of nodes to the entry \([\text{its maximum distance in terms of events to the entry}]\).

\( x_5 \): The number of nodes (events) of a sub-node as sub-menus that can be reached from this node (maximum number of sub-functions that can be invoked further).

\( x_6 \): The total number of occurrences of an event (a node) within all CESs, i.e., walks (significance of an event).

\( x_7 \): The balancing degree determines balancing a node as the sum of all incoming edges (as plus \((+))\) and outgoing edges (as minus \((-))\) for a given node.

\( x_8 \): The averaged frequencies of the usage of events within the CESs (determines the averaged occurrence of each event within all CESs).

\( x_9 \): The number of FEPs connected to the node under consideration (takes the number of all potential faulty events entailed by the event given into account).

4.2. Definition of Importance Degree and Preference
The CESs are manually ordered, scaling their preference degrees based on the events which incorporate the importance group(s). Importance \((\text{Imp}(e))\) of \(e^{th}\) event is defined as follows\(^{28}\):

\[
\text{Imp}(e) = c - \text{ImpD}(S_k) + 1 \quad (22)
\]

where \(c\) is the optimal number of the groups; \(\text{ImpD}(S_k)\) is defined by means of the importance degree of the group \(S_k\) to which the \(e^{th}\) event belongs.
Finally, choosing the events from the ordered groups, a ranking of CESs is formed according to their descending preference degrees. The assignment of preference degrees to CESs is based on the following rule:

(a) The CES under consideration has the highest degree if it contains the events which belong to the “top” group(s) with utmost importance degrees, i.e., that is placed within the highest part of group ordering.
(b) The CES under consideration has the lowest degree if it contains the events which belong to the group(s) that are within the lowest part of the “bottom” group(s) with least importance degree i.e., that is placed within the lowest part of group ordering.

Therefore, the preference degree of CES can be defined by taking into account both the importance of events (22) and the frequency of occurrence of event(s) within them that is formulated as follows:

\[
\text{Pref}(\text{CES}_q) = \sum_{e=1}^{n} \text{Imp}(e) f_q(e) \quad q = 1, \ldots, m \in \mathbb{N}
\]  

where \(m\) is the number of CESs, \(n\) is the number of events, \(\text{Imp}(e)\) is importance degree of the \(e^{th}\) event (22) and \(f_q(e)\) is frequency of occurrence of event \(e\) within CES\(_q\). This order determines the preference degree \(\text{Pref}(\text{CES}_q)\) of CESs as test cases.

**Indirect Determination of the Preference Degree**

**Step 1.** Construction of a set of events \(X = \{x_{ij}\}\) where \(i = 1, \ldots, n \in \mathbb{N}\) is an event index, and \(j = 1, \ldots, p \in \mathbb{N}\) is an attribute index.

**Step 2.** Training the NN using adaptive CL algorithm (see Section 3.2).

**Step 3.** Classification of the events into \(c\) groups ((21), see Section 3.2).

**Step 4.** Determination of importance degrees of groups according to length \((\ell)\) of weight vectors.

**Step 5.** Determination of importance degrees of event groups ((22), this present section).

**Step 6.** An ordering of the CESs for prioritizing the test process.

### 5. A Case Study

Based on the web-based system ISELTA (*Isik’s System for Enterprise-Level Web-Centric Tourist Applications*), we now present a case study to validate the testing approach presented in the previous sections. Both the construction of ESGs, and generation of test cases from those ESGs, have
been explained in the previous papers of the first author.\textsuperscript{9−11} Therefore, the case study concentrates on the test prioritizing problem.

ISELTA has been developed by our group in cooperation with a commercial enterprise to market various tourist services for traveling, recreation and vacation. It can be used by hotel owners, travel agents, etc., but also by end consumers. A screenshot in Fig. 2 demonstrates how to define and reserve rooms of different types.

5.1. Derivation of the Test Cases

Figure 3 depicts the completed ESG of the scenario described above and in Fig. 2. Test cases can now be generated using the algorithms mentioned in Section 3 and described in\textsuperscript{10−11} in detail. For the lack of space, reference is made to these papers and the CESs generated are listed in Table 1.
Fig. 3. Completed ESG for room definition/selection (solid arcs: event pairs (EP); dashed).

Legend of Fig. 3:

1: Click on “Starting”
2: Click on “Registering”
3: Registration carried out
4: Click on “log in”
5: Logged in
6: Click on “Password forgotten”
7: Password forgotten
8: Click on “Request”
9: Indicate service(s) offered
10: Indicate administrator
11: Indicate agent

Table 1. CESs of ESG in Fig 3.

<table>
<thead>
<tr>
<th>CES1</th>
<th>CES2</th>
<th>CES3</th>
<th>CES4</th>
<th>CES5</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4 5 4 5 9 1 4 5 10 1 4 5 11 1 4 5 9 2 3 4 5 10 2 3 2 3 1 4 5 11 2 3 4 6 4 6 7 8 1 2 3]</td>
<td>[1 4 5 9]</td>
<td>[2 3 4 6 7 8 2 3 4 5 10]</td>
<td>[4 5 11]</td>
<td>[4 6 7 8]</td>
</tr>
</tbody>
</table>

5.2. Determination of Attributes of Events

As a follow-on step, each event, i.e., the corresponding node in the ESG, is represented as a multidimensional data point using the values of all nine attributes as defined in the previous section. Estimating by means of the
Table 2. Data set of events.

<table>
<thead>
<tr>
<th>Event No</th>
<th>X_1</th>
<th>X_2</th>
<th>X_3</th>
<th>X_4</th>
<th>X_5</th>
<th>X_6</th>
<th>X_7</th>
<th>X_8</th>
<th>X_9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>39</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>0.1860</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>40</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>0.1519</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>41</td>
<td>6</td>
<td>7</td>
<td>−3</td>
<td>0.1519</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>7</td>
<td>10</td>
<td>35</td>
<td>0</td>
<td>14</td>
<td>3</td>
<td>0.2469</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>29</td>
<td>0</td>
<td>10</td>
<td>−3</td>
<td>0.2112</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td>36</td>
<td>0</td>
<td>4</td>
<td>−1</td>
<td>0.2199</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>37</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0.1218</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>10</td>
<td>38</td>
<td>0</td>
<td>3</td>
<td>−2</td>
<td>0.1218</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
<td>10</td>
<td>17</td>
<td>30</td>
<td>3</td>
<td>−2</td>
<td>0.1494</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>4</td>
<td>10</td>
<td>22</td>
<td>0</td>
<td>3</td>
<td>−2</td>
<td>0.0698</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>4</td>
<td>10</td>
<td>30</td>
<td>0</td>
<td>3</td>
<td>−2</td>
<td>0.1911</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 3. Obtained groups of events.

<table>
<thead>
<tr>
<th>Groups (3) and (13)</th>
<th>Events</th>
<th>Length of weight vectors (ℓ)</th>
<th>Importance Degree</th>
<th>ImpD(S_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1</td>
<td>9</td>
<td>2.07</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>S_2</td>
<td>3,6,7,8,11</td>
<td>2.04</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>S_3</td>
<td>1,2,4</td>
<td>1.97</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>S_4</td>
<td>5</td>
<td>2.25</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>S_5</td>
<td>10</td>
<td>1.73</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

ESG and ESG, the values of attributes for all events are determined and the data set \( X = \{ x_1, \ldots, x_{11} \} \subset \mathbb{R}^9 \) is constructed as in Table 2.

5.3. Construction of the Groups of Events

For the data set gained from the case study (Fig. 2 and 3), the optimal number \( c \) of the groups is determined to be 5 which leads to the groups \( S_k, k = 1, \ldots, 5 \). Importance degrees (\( \text{ImpD}(S_k) \)) of obtained groups are determined by comparing the length of their weight vectors (\( ℓ \)) and all \( \text{ImpD}(S_k) \) values that are presented in Table 3.

5.4. Indirect Determination of Preference Degrees

As mentioned in the previous section, the preference degree of the CESs is determined indirectly by (23) that depend on the importance of events (22) and frequency of event(s) within CES. The ranking of the CESs is represented in Table 4.
Table 4. Ranking of CESs (walks).

<table>
<thead>
<tr>
<th>PrefDeg.</th>
<th>Pref (CES(e))(15)</th>
<th>CESs</th>
<th>CESs (walks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>126</td>
<td>CES(_1)</td>
<td>[4 5 4 5 9 1 4 5 10 1 4 5 11 1 4 5 9 2 3 4 5 10 2 3 2 3 1 4 5 11 2 3 4 6 4 6 7 8 1 2 3]</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>CES(_3)</td>
<td>[2 3 4 6 7 8 2 3 4 5 10]</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>CES(_2)</td>
<td>[1 4 5 9]</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>CES(_5)</td>
<td>[4 6 7 8]</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>CES(_4)</td>
<td>[4 5 11]</td>
</tr>
</tbody>
</table>

Exercising the test cases (CESs, or walks) in this order ensure that the most important tests will be carried out first. Moreover, the achieved ranking of CESs complies with the tester’s view. Thus, an ordering of the complete set of CESs (walks) is determined using the test suite generated by the test process, i.e., we now have a ranking of test cases to make the decision of which test cases are to be primarily tested. Undesirable events can be handled in a similar way; therefore, we skip the construction of ranking of the FCES.

6. Conclusions and Future Work

The model-based, coverage-and specification-oriented approach described in the previous sections provides a novel and effective algorithm for ordering the test cases according to their degree of preference. Such degrees are determined indirectly through the use of the events specified by several attributes, and not a single one. This is an important issue and consequently, the approach introduced radically differs from the existing ones.

The relevant attributes are visualized by means of a graphical representation (here, given as a set of ESGs). The events (nodes of ESG) are classified using unsupervised neural network clustering. The approach is useful when an ordering of the tests due to restricted budget and time is required. Run-time complexity of this approach is of \(O(n^2)\), assuming that the number of events \(n\) greater than the number of attributes \(p\), otherwise it is \(O(p^2)\).

We plan to apply our prioritization approach to a more general class of testing problems, e.g., to multiple-metrics-based testing where a family of software measures is used to generate tests. Generally speaking, the
approach can be applied to prioritize the testing process if the SUT is modeled by a graph of the nodes which represent events or sub-systems of various granularities (modules and functions, or objects, methods, classes, etc.).

References
Prioritizing Coverage-Oriented Testing Process