Test case generation for transition-pair coverage using Scatter Search

Raquel Blanco, José García-Fanjul, Javier Tuya
Computer Science Department, University of Oviedo
Campus Universitario de Gijón s/n, Gijón, SPAIN
rblanco@uniovi.es, jgfanjul@uniovi.es, tuya@uniovi.es

Abstract

A challenging part of Software Testing entails the generation of test cases, whose costs can be reduced by means of the use of techniques for automating this task. On the other hand, the nature of Software Engineering problems is ideal for the application of metaheuristic techniques. In this paper we present an approach based on the metaheuristic technique Scatter Search for the automatic test case generation of BPEL business processes using a transition-pair coverage criterion. The test case generator is called TCSS-LS-for-BPEL and it combines a diversity property with a local search. The diversity property is used to extend the search of test cases in order to reach different transitions of the business process. The local search is used to intensify the search when the diversification has problems to find test cases that cover transitions that have not been covered yet. We present the results obtained by our test case generator using two sample compositions and carry out a comparison with a random generator. The results indicate that TCSS-LS-for-BPEL can be used in the generation of test cases for BPEL business processes.

Keywords: *software testing, automatic test case generation, BPEL web service compositions, Scatter Search, transition-pair coverage.*

1. Introduction

Testing is a very important, though expensive, phase in software development and maintenance and a challenging part of this phase entails the generation of test cases. This generation is crucial to the success of the test because a suitable design of test cases will be able to detect a great number of faults. Furthermore, the generation of test cases is perhaps the most expensive task in software testing, since this process is mainly manual, and it can involve approximately 40% of the total cost of software testing [32]. This cost can be reduced by means of the use of techniques for automating the generation of test cases. In service oriented architectures the deployment of software as a service has the objective that, in the short or medium term, these services will be invoked from other software or services. To describe the interaction among the services, the BPEL language is commonly used. Thus, using well-established and automated testing techniques is essential to assure the quality of the deployed services and also to facilitate regression testing.

The search for an optimal solution in the test case generation problem has a great computational cost and for this reason the techniques for automating the generation of test cases try to obtain near optimal solutions. As a consequence, they have attracted growing interest from many researchers in recent years. On the other hand, the nature of Software Engineering problems is ideal for the application of metaheuristic techniques, as is shown in the work of Harman and Jones [17]. One such problem is software testing, which is treated as

a search or optimization problem, as is shown in several reviews [21][22]. Moreover, the metaheuristic techniques have obtained good results in test case generation [21].

This work proposes the use of the metaheuristic technique Scatter Search [16][18] to generate test cases for BPEL business processes using a transition-pair coverage criterion. The approach presented is called TCSS-LS-for-BPEL and it is an evolution of the algorithm TCSS-LS described in [6], which generates test cases for the branch coverage criterion for programs written in C, and the algorithm presented in [4], which generates test cases for BPEL business processes using a transition coverage adequacy criterion.

The rest of the paper is organized as follows. The following section presents a brief description of BPEL business processes, related work and the Scatter Search technique. Section 3 presents the problem representation for transition-pair coverage for BPEL business processes. Section 4 details our Scatter Search approach for the automatic generation of test cases. In Section 5 we present the results and conclusions are presented in Section 6.

2. Background

In this section we briefly describe the specification of web service compositions using BPEL, present related work and explain the Scatter Search technique.

2.1. BPEL business processes

BPEL specifications represent the behavior of business processes based on web service compositions. They are XML documents composed of two main sections: declarations and the specification of the business process itself. In the declarations part, partnerlinks and portTypes are identified: each partnerlink stands for a service that interacts with the business process and portTypes define the details of the interfaces between services and the business process. Other elements included in this first part are the variables, which enable the intermediate storage of values.

The specification of the business process consists of a set of activities that can be executed. These activities may be either basic or structured. Among the former, the business process can invoke web services or receive invocations by means of the invoke and receive activities. It can also update the values of the variables using assign. Structured activities prescribe the order in which a collection of activities takes place. For example: a sequence activity establishes a sequential order and a while forces the repetition of the execution of a set of activities until a given condition becomes false. Another kind of activity is the flow, which groups concurrent activities.

An extract of the sample BPEL business process called “loan approval” is outlined in Figure 1. This example was published within the specification of the standard [26]. The goal of this business process is to conclude whether a certain request for a loan will be approved or not. To do so, it receives a request from a partner called “customer” and invokes two other partners. The “assessor” partner measures the risk associated with low amount requests. Another partner, called “approver”, approves requests that are either made for a large amount of money or which are evaluated by the assessor as not having a low risk.

In order to generate test cases for a BPEL business process, adequacy criteria such as transition coverage and transition-pair coverage can be used. A transition T_i from an activity A_m to an activity A_n indicates that A_n is executed just after A_m. A transition-pair P_{ij} is defined by means of two transitions T_i, T_j that are executed consecutively. To fulfill the transition coverage criterion all transitions T_i of the business process must be covered and to fulfill the

transition-pair coverage criterion all transition-pairs P_{ij} must be reached for the test cases generated.

```
<process name="loanapproval" [...]>
  <-- declarations -->
  <variables>
    <variable name="riskAssessment"
              messageType=
        "asns:riskAssessmentMessage"/>
  [...]
  </variables>
  <partners>
    <partner  name="customer" [...]/>
    <partner name="assessor" [...]/>
    <partner name="approver" [...]/>
  </partners>
  <-- behaviour of the business process -->
  <flow>
    <links>
      <link name="receive-to-assess"/>
      <link name="assess-to-setMessage"/>
      [...]
    </links>
    <receive name="receive1"
      partner="customer" [...]>
      [...]
    </receive>
    <invoke name="invokeAssessor"
         partner="assessor"
         portType="asns:riskAssessmentPT"
         operation="check"
         inputVariable="request"
         outputVariable="riskAssessment">
      <target linkName="receive-to-assess"/>
      <source linkName="assess-to-setMessage"
          transitionCondition=
          "bpws:getVariableData
          ('riskAssessment','risk') ='low'"/>
      <source linkName="assess-to-approval"
          transitionCondition="
          bpws:getVariableData
          ('riskAssessment','risk') !='low'"/>
    </invoke> [...]
  </flow>
</process>
```

Figure 1. Extract from the "loan approval" BPEL specification

2.2. Metaheuristic techniques for test case generation

The most widely used metaheuristic technique in test case generation is Genetic Algorithms. This technique is used in many papers to achieve several coverage criteria [1][2][15][25][31][32]. Other papers apply Genetic Algorithms to generate test cases to cover string predicates [3], to detect overflows [9], for regression test case prioritization [19], to train a series of decision trees in order to create rules for classifying test cases [30] and to generate test data that cause service level agreement violations in service-oriented systems [10].

Other metaheuristic techniques are also applied in the generation of test cases, such as simulated annealing, genetic programming or tabu search. Simulated annealing has been used

to generate test cases to achieve several coverage criteria [20][32] and it has been used in the investigation of measures of landscape to apply this technique to test generation [29]. Genetic Programming has been used in the classification task in the context of data mining of relational databases and the selection of test cases using the mutation testing adequacy criterion in the context of software testing [28]. Tabu Search has been used to obtain branch coverage [12] and path and loop coverage [11]. Simulated Repulsion has been used to generate diverse test data and evaluate the effect of diversity on data flow coverage and mutation testing [7]. Evolutionary algorithms have been used in the automation of functional testing [8], and their principles have been combined with an extended chaining approach to find test cases that cover a target [23]. Hill Climbing has been used in the regression test case prioritization [19]. Evolutionary Strategies have been used to achieve condition coverage [2]. Estimation of Distribution Algorithms has been used to obtain branch coverage [27]. Scatter Search has been used to reach branch coverage [5][6][27].

Regarding the generation of test cases for BPEL business processes, approaches rely on techniques that derive test cases from a specification of the expected behavior of the software under test. A method to test BPEL business processes using model checking is shown in [14], which guides the selection of test cases to reach transition coverage. High level Petri Nets have been also used to model BPEL business processes, applying existing tools to analyze these models and generate test cases [13]. Other approaches use different formalisms to obtain the test cases from the BPEL specification. A data-flow technique is combined with term-rewriting tools to obtain test cases for BPEL processes [24]. Another work prescribes a control-flow method and expounds how to generate tests from a model of the flow of BPEL activities and using a constraint solver [33]. The use of metaheuristic techniques in this problem is very recent and the only work that applies one of these techniques to test BPEL business processes is our previous work [4], which generates test cases to fulfill a transition coverage criterion using the Scatter Search technique.

2.3. Scatter Search technique

Scatter Search [16][18] is an evolutionary method that works on a population of solutions of the problem to be solved, which are stored in a set of solutions called the Reference Set. The solutions in this set are combined in order to obtain new ones, trying to continually generate better solutions, according to quality and diversity criteria.

The basic scheme of the Scatter Search algorithm can be seen in Figure 2 [18]. The Scatter Search algorithm begins by using a diversity generation method to generate P diverse solutions, to which an improvement method is applied. Then the Reference Set is created with the best solutions from P and the most diverse in relation to the solutions already in the Reference Set. As new solutions are generated, the algorithm produces subsets of the Reference Set using a subset generation method, and applies a solution combination method in order to obtain new solutions, to which an improvement method is applied. Then a Reference Set update method evaluates the new solutions to verify whether they can update the Reference Set, as they are better than some solutions stored in the set. If so, the best solutions are included in the Reference Set and the worst solutions are dropped. So, the final solution of the problem to solve is stored in the Reference Set.

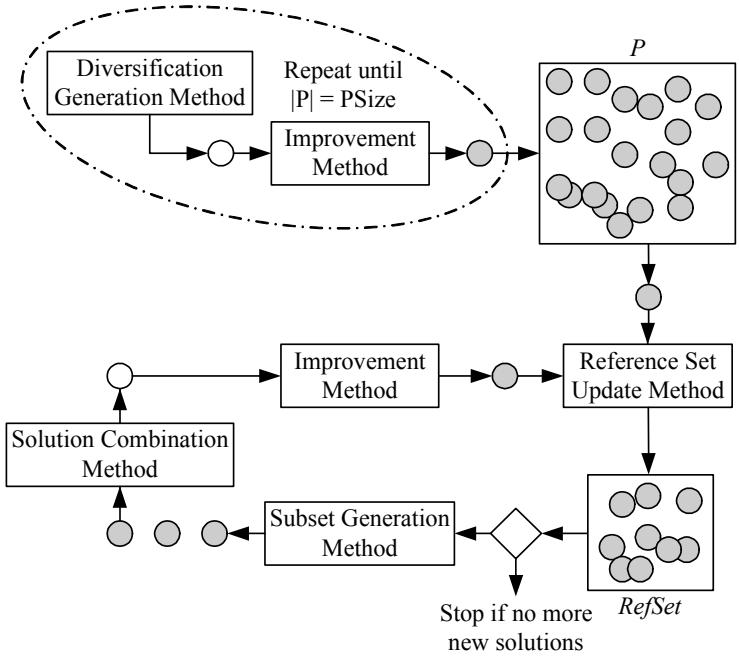

Figure 2. Basic scheme of Scatter Search

3. Problem representation

This section describes our representation of the transition-pair coverage criterion, which is based on several transformations of the state graph that represents the business process for handling path forks, loops and faultHandlers activities.

3.1. Coverage criteria for transitions

The BPEL business process can be represented as a state graph, as shown in [4], where the nodes represent the states of the business process and the arcs represent the transitions in the business process, i.e., the change of state from node i to node j when the associated arc decision is true.

As our goal is to generate test cases that allow all transition-pairs of the business process to be covered, our approach transforms the state graph that represents it in order to obtain a graph in which each arc corresponds to a transition-pair. This state graph is called transition-pair graph. By means of this new state graph, it is possible to determine the transition-pairs covered by the test cases generated, since the business process has been instrumented to know the followed path.

The main idea of the transformations consists of joining two consecutive transitions to obtain a new one. The decision of this new arc is formed by the conjunction of the decisions of the transitions joined. Furthermore, a new state is also created to represent the states of the original graph that are reached when the transitions to be joined are executed. Figure 3 shows the basic idea of the transformations. State A has two input transitions T_1 and T_2 and two output transitions T_3 and T_4. In order to generate the transition-pairs that appear in the right part of the figure we need to combine the input transitions with the output transitions. So four pairs are formed: P_{13} which represents the transition-pair that joins transitions T_1 and T_3 (the decision of the arc P_{13} is T_1 && T_2), P_{23} (joins transitions T_2 and T_3), P_{14} (joins transitions T_1

and T_4) and P_{24} (joins transitions T_2 and T_4). Regarding the states, state B' signifies that the business process is in state B and its previous state was state A and state C' indicates that the business process reaches state C just before reaching state A. Now state B' has two input arcs P_{13} and P_{23} because these pairs include the transition T_3 that is the input transition of state B. In the same way, state C' also has two input arcs P_{14} and P_{24}.

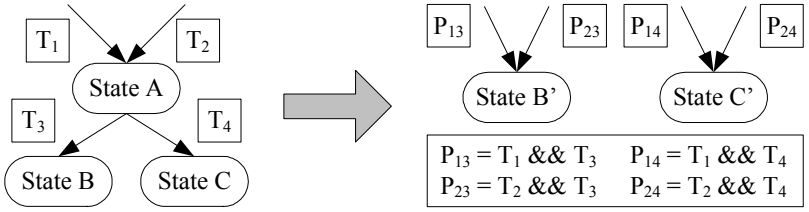

Figure 3. Basic transformation

The transition-pair coverage criterion generates more test cases than the transition coverage criterion, as can be seen in the previous example. When a transition-pair coverage criterion is used to generate test cases for this example, we need to find four test cases (a test case that covers each pair), whereas to fulfill the transition coverage criterion only two test cases are needed (for instance, one test case can reach transitions T_1 and T_3 and another test case can cover transitions T_2 and T_4).

3.2. Transformation for path forks

We illustrate the transformation for path forks by means of the graphs of Figure 4. In the left part of the figure we can see a state graph with two different types of forks: a nested fork (fork below state F) and a fork after the union of a previous fork (fork below state D).

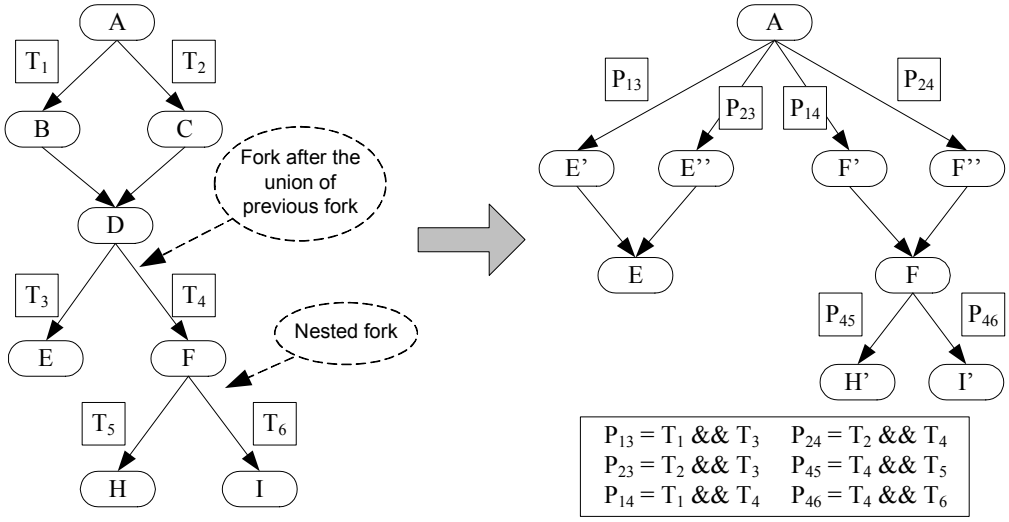

Figure 4. Transformation for path forks

A nested fork has a transition T_k that ends in a state A_h and n transitions T_i (i=1..n) that start in this state A_h. The transition-pair graph has n arcs P_{ki} and each one is formed by the union of T_k and a specific T_i. For instance, state F in the left part of Figure 4 has the input transition T_4 and the output transitions T_5 and T_6. The transition-pair graph that is shown is the right part of Figure 4 has a state F with two output arcs, P_{45} and P_{46}, which correspond to

the union of T_4-T_5 and T_4-T_6 respectively. The state H' indicates that the business process has reached the state H after executing the transitions T_4 and T_5 consecutively. In the same way, state I' indicates the business process is in state I after executing the transitions T_4 and T_5 consecutively.

A fork after the union of a previous fork has n transitions T_i (i=1..n) that end in a same state A_h and m transitions T_j (j=1..m) that start in the state A_h. The transition-pair graph has n·m arcs P_{ij} and each one is formed by the union of a specific T_i and a specific T_j. A special situation occurs when a transition T_i does not have an associated decision. In that case T_i is substituted by the q transitions T_o (o=1..q) that end in its starting state, which have an associated decision. In that case the transition-pair graph has q·m arcs P_{oj} for the combination of that T_i and the transitions T_j, where each arc if formed by the union of a specific T_o and a specific T_j.

For instance, the left part of Figure 4 has a state D that joins a previous fork and has two output transitions (T_3 and T_4). As the input transitions of state D do not have an associated decision, those transitions are substituted by T_1 and T_2 to form the transition-pairs. The transition-pair graph that is shown in the right part of Figure 4 has four arcs that are generated by means of the combination of the input transition T_1, T_2 and the output transitions T_3, T_4: $P_{13} = T_1$ and T_3, $P_{23} = T_2$ and T_3, $P_{14} = T_1$ and T_4, $P_{24} = T_2$ and T_4. State E' indicates that the business process has reached the state E after achieving state B, that is, the transitions T_1 and T_3 have been executed consecutively. State E'' also indicates the business process is in state E, but they differ from the previous state. In this case, the previous state was state C, that is, the transitions T_2 and T_3 have been executed consecutively. As both states E' and E'' represent that the business process has reached the state E they are joined in this state. In the same way, states F' and F'' indicate that the business process has achieved the state F, although they differ from the previous state reached, and therefore they are joined in this state.

3.3. Transformation for loops

When a loop appears in the business process, we need to check the following transition-pairs, as can be seen in Figure 5: a pair that joins the transition before the loop and the output transition of the loop, a pair that joins the transition before the loop and the first transition inside the loop, and two pairs that join the transition that represents the loop feedback with the first transition of the loop and the output transition of the loop respectively. With these pairs, it is possible to test if a loop is not executed, a loop is executed once and a loop is executed several times. These situations are also checked in a loop coverage criterion. In order to verify the aforementioned pairs two states that represent switch activities are included in the transition-pair graph.

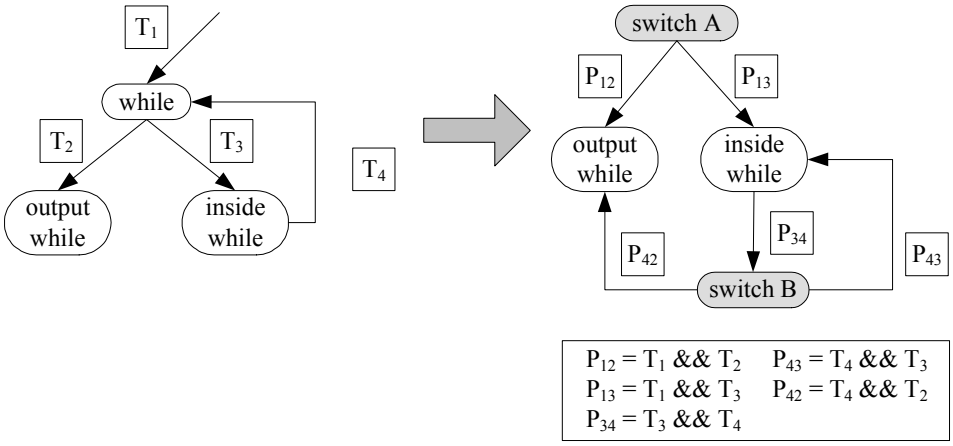

Figure 5. Transformations for loops

Figure 5 shows the transformation for loops. Both switch states handle the same decision used by the while state of the left part of the figure. Switch A is used to check the pairs P_{12} (it joins T_1 and T_2 and indicates that the loop is not executed) and P_{13} (it joins T_1 and T_3 and indicates that the loop is executed at least once). Switch B is used to check the pairs P_{42} (it joins T_4 and T_2 and represents that the loop is not executed any more) and P_{43} (it joins T_3 and T_3 and represents that the loop is executed one more time).

3.4. Transformation for faultHandlers

The transformation for faultHandlers follows the same outline of the forks transformation. In this case, we need to include instrumentation in the business process to remember the last invoke activity that has been executed. The instrumentation is carried out in each catch block of the faultHandlers activity, as is shown in Figure 6. Each catch block includes a switch activity that is used to check the invoke activity that raises the exception handled by this catch block and therefore to recognize the transition-pair reached.

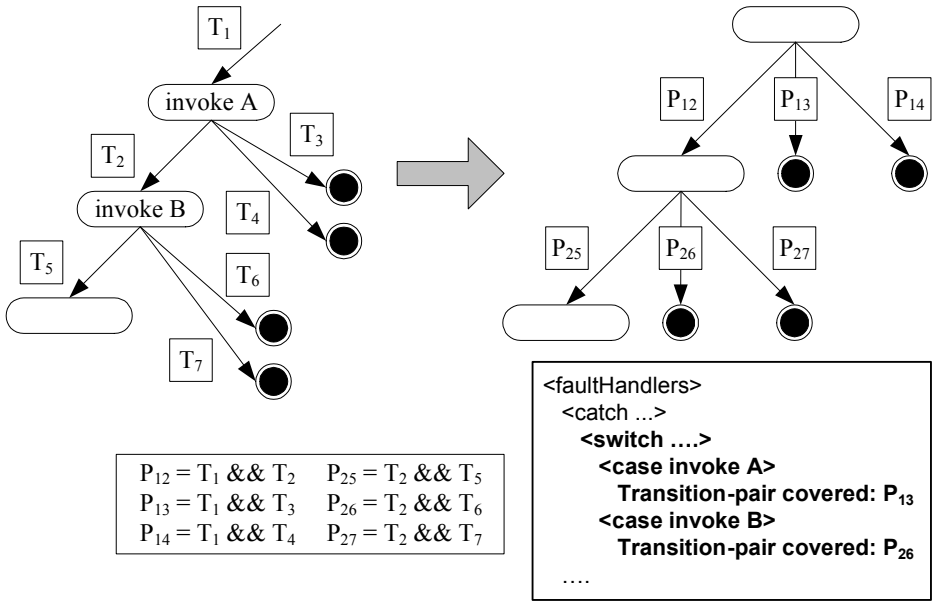

Figure 6. Transformation for faultHandlers

4. Test case generation using Scatter Search

This section describes the adaptation of the Scatter Search technique, called TCSS-LS-for-BPEL, and the use of the transition-pair graph to automatically generate test cases for BPEL business processes using a transition-pair coverage criterion. To define a test case we need the input variables of the business process and the transition-pairs that are executed. The input variables are the variables received from the web services (called partners in BPEL) that interact with the business process. By means of the sequence of transition-pairs covered we can determine the order in which the partners have given the values of the input variables to the business process. This order is important to generate the test cases, because two sequences of values of the input variables with a different order can cover different transition-pairs.

The general goal that consists of generating test cases that allow all transition-pairs of the business process to be covered is divided into subgoals, each of which consists in finding test cases that reach a particular arc (transition-pair) P_{ij} of the transition-pair graph.

In order to reach the subgoals, the arcs of the transition-pair graph store information during the process of test case generation. This information is used to know the transition-pairs covered and to make progress in the search process. Each arc stores this information in its own set of solutions, called Reference Set. Unlike the original Scatter Search algorithm, our approach has several Reference Sets. Each Reference Set is called S_{ij}, where ij represents a transition-pair of the transition-pair graph, and is formed by B_{ij} elements $T_{ij}^c = <\bar{x}_{ij}^c, p_{ij}^c, \overline{fb}_{ij}^c, \overline{fc}_{ij}^c>$, $c \in \{1..B_{ij}\}$, where:

- $\bar{x}_{ij}^c$ is a solution, i.e., a test case that reaches arc P_{ij}. Each solution $\bar{x}_{ij}^c$ consists of a set of given values for the input variables ($\bar{x}1$, $\bar{x}2$,..., $\bar{x}n$) of the business process under test that satisfy the transition-pairs represented by the arcs which form the path that has been followed to arc P_{ij}. Each input variable is related to a web service and is represented as a vector since the web service can be invoked several times and each invocation provides an independent value.

- p_{ij}^{c} is the path covered by the solution (test case), i.e., the sequence of the arcs of the transition-pair graph reached by the solution.
- $\overline{fb}_{ij}^{c}$ is the vector of distances to the sibling arcs. These distances indicate how close the solution came to cover the sibling arcs, i.e., the sibling transition-pairs.
- $\overline{fc}_{ij}^{c}$ is the vector of distances to the next arcs that has not been reached by the solution. These distances indicate how close the solution came to cover these arcs.

The distances are calculated using the decisions of the arcs of the transition-pair graph that have not been reached during the execution of the business process, i.e., the false decisions. The function used to calculate the distances can be consulted in [6].

An example of the transition-pair graph with the information stored in the Reference Sets of the arcs can be seen in Figure 7.

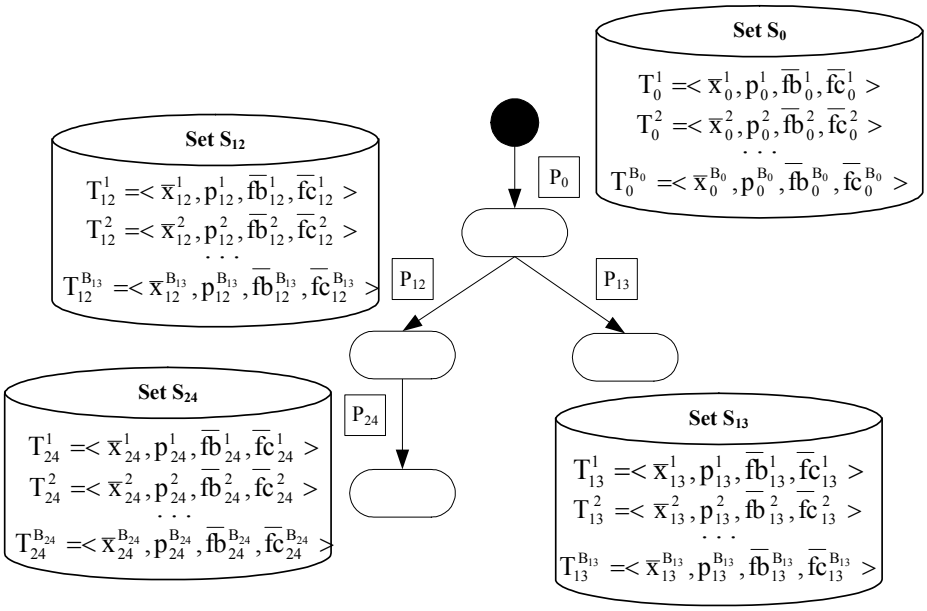

Figure 7. TCSS-LS-for-BPEL transition-pair graph

Each set S_{ij} has a different size B_{ij} that depends on the complexity of the business process situated below the transition-pair P_{ij}. The procedure followed to calculate the maximum size B_k can be consulted in [6]

TCSS-LS-for-BPEL will try to make the sets as diverse as possible in order to generate solutions that can cover different transition-pairs of the business process. The diversity of a solution of a set S_{ij} is a measure related to the path covered by all solutions of the set.

4.1. Search process

The goal of TCSS-LS-for-BPEL is to obtain maximum transition-pair coverage, i.e., to find solutions that cover all arcs of the transition-pair graph. As these solutions are stored in the sets S_{ij}, the goal is therefore that all the sets have at least one element. If the composition under test has unfeasible transitions-pairs, the goal cannot be reached, so TCSS-LS-for-BPEL also stops its execution when a maximum number of test cases has been generated.

As the BPEL specification does not directly include information about the behavior of the different web services that participate in the business process, a mock model will be

constructed for each partner based upon its interface with the business process, in order to carry out the search process.

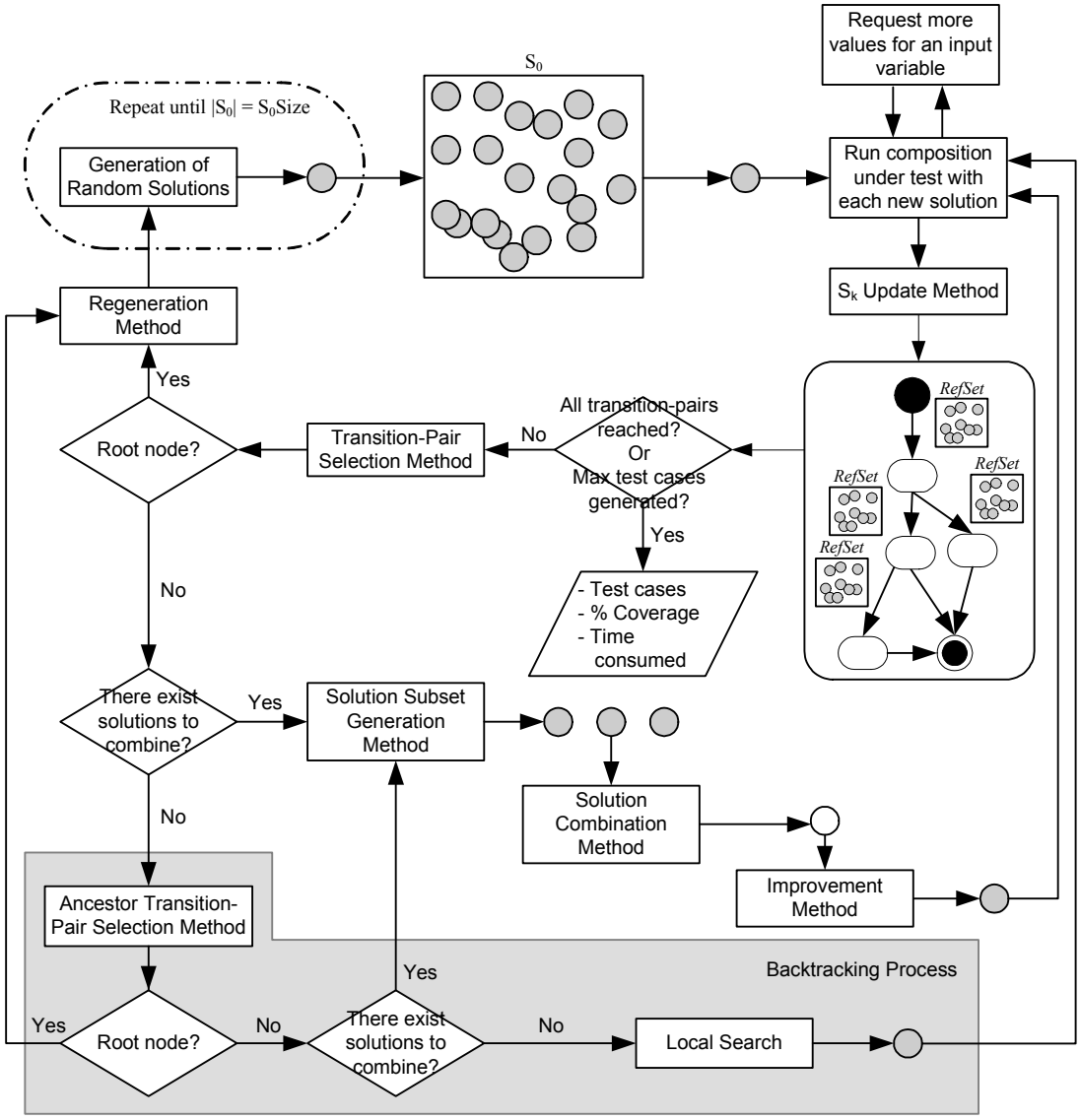

Figure 8. TCSS-LS-for-BPEL scheme

Figure 8 shows the scheme of the search process. The first step consists of generating random solutions which are stored in the set S_0 (set of arc P_0 that represents the starting point). The service composition model is executed with each solution and the sets S_k of the arcs reached are updated. Then, the iterations of the search process begin and TCSS-LS-for-BPEL selects in each one an arc to form the subsets of solutions from its set S_{ij}. These subsets are used by the combination rules to generate the new solutions, which can be improved. The new solutions are executed in the service composition model in order to update the set S_{ij} of the arcs achieved and the cycle of execution is closed.

Every time the composition model is executed, the partners must be configured with the values of the variables they returned to the business process when they are invoked, i.e., the partners are configured with the solution to be executed in the model. This solution may not have enough values for a specific variable, since the invoke activity that returns it to the business process can be inside a loop and this loop can be executed an unknown number of times. When a partner does not have enough values for the variable it returns, it must ask TCSS-LS-for-BPEL for the new values for the variable.

On the other hand, if the set S_{ij} of the arc selected by TCSS-LS-for-BPEL does not have at least two solutions that can be used by the combination rules to generate new solutions, a backtracking process is carried out. This backtracking combines the Scatter Search technique with a Local Search method.

The backtracking process, the combination rules, and the methods carried out by TCSS-LS-for-BPEL can be consulted in [6].

The search process finishes when all transition-pairs have been covered or the maximum number of test cases has been generated.

The final solution of TCSS-LS-for-BPEL consists of the test cases that cover the transition-pairs, which are stored in the sets S_{ij}, the percentage of transition-pair coverage reached and the time consumed in the search process.

4.2. History of values of a variable

The business process can include a web service invocation inside a loop, and therefore the variable returned by the partner has a different value in each loop iteration. According to the different sequences of values of the variable that the partner can return in the loop iterations different transition-pairs of the business process can be covered. For that reason all values of the variable and the order in which they are returned by the partner must be considered to generate the test cases. As the number of iterations of a loop is often unknown, an input variable can take an unknown number of values in the execution of a composition and the vector $\overline{x}h$ that represents it in the solution $\overline{x}_k^c$ can have a different size in two specific solutions. Moreover, the number of iterations of the loop can depend on the values of the variable returned by the partner inside the loop. For instance, Figure 9 shows a loop whose decision depends on the values of the variable returned by the partner. In this example, the partner is invoked inside the loop and returns a value for the variable *item* in each iteration of it, which is accumulated in the variable *itemSum*. The loop is executed until the sum of values of the variable *item* is greater than 10 (*itemSum > 10*). A possible test case can have the sequence of values 6, 2, 5 for the variable item. Note that the order in which the partner returns the values of the variable is important, since if it returns the value 5 after the value 6, the loop finishes its execution and the value 2 is not necessary.

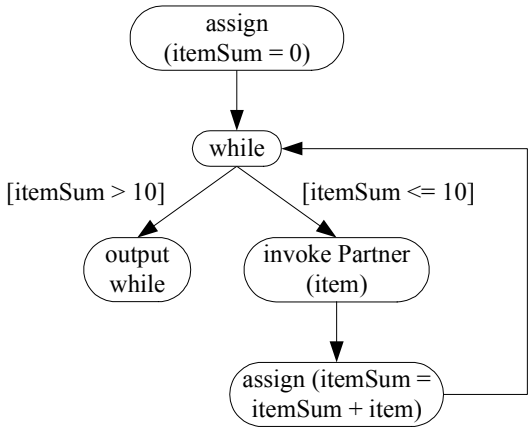

Figure 9. Loop example

For that reason the algorithm described in [6] has been improved to include a new method to handle the unknown and different number of values of an input variable.

First, TCSS-LS-for-BPEL generates random solutions that are represented by means of a vector for each input variable. TCSS-LS-for-BPEL constructs the vectors with a specific number of values (all variables have the same number of values). Then the partners of the business process are configured with the vectors of the variables they returned and the business process is executed. When a partner is invoked it returns a value of the vector of the variable and when it has used all values of the variable it ask TCSS-LS-for-BPEL for the new values.

TCSS-LS-for-BPEL searches the new values for an input variable among the solutions of the set S_{ij} of the arc that is used to generate the new solutions. The algorithm tries to find the most diverse values for the input variable. Thus, the function "diversity of a variable" is defined. TCSS-LS-for-BPEL applies this function over the subset $S'_{ij}=\{T'^{1}_{ij},\ldots,T'^{q}_{ij}\} \subseteq S_{ij}$, $T'^{c}_{ij}=<\bar{x}'^{c}_{ij};p'_{ij};\overline{fb}'^{c}_{ij};\overline{fc}'^{c}_{ij}>$, which represents the solutions stored in the set S_{ij} that have not been used to give new values to the partner. The diversity value of a variable $\bar{x}h$ is calculated according to the function defined as:

$$div\_var(<\bar{x}'^{m}_{ij}>;S'_{ij}) = \sum_{c=1..q}\left(\sum_{z=1..r}\begin{cases} \left|\bar{x}'^{m_z}_{ij}\right| & \text{if size of } \bar{x}'^{c}_{ij} < z \\ \left|\bar{x}'^{c_z}_{ij}\right| & \text{if size of } \bar{x}'^{m}_{ij} < z \\ \left|\bar{x}'^{m_z}_{ij} - \bar{x}'^{c_z}_{ij}\right| & \text{otherwise} \end{cases}\right)$$

where index c = 1..q covers the Solutions of the set S'_{ij} and the index z=1..r covers the values of the vector that represents the input variable $\bar{x}'$.

TCSS-LS-for-BPEL gives the partner the values of the variable $\bar{x}'$ of the solution with the highest value of *div_var()*, as that solution is the least similar to the rest of the solutions according to the values for the variable $\bar{x}'$. Thus, the size of the vector of values that represents the variable $\bar{x}'$ in the solution that is executed in the business process is increased.

When the business process finishes its execution, TCSS-LS-for-BPEL analyzes the solution in order to drop the values of the variables that have not been used. Thus, the size of the vectors is decreased. Then the updating process of the sets S_{ij} is carried out and they will store solutions with different sizes for the vector of values of a specific variable.

On the other hand, the solution combination method, the local search method and the update method presented in [6] have also been adapted to handle the different number of values of the input variables in several solutions, as described in [4].

5. Case studies

The algorithm TCSS-LS-for-BPEL and the transformations presented in this work have been applied to two BPEL specifications: "loan approval" and "shipping service". Both specifications were originally published within the standard BPEL4WS and have been extensively referenced in the literature on web services testing. The "loan approval" has been described above in section 2.1. The "shipping service" composition describes a basic shipping service that handles the shipment of orders. It offers two types of shipments: shipments where the items are held and shipped together and shipments where the items are shipped piecemeal until all of the order is accounted for. In order to check the methods designed in our approach we have modified the "shipping service" composition as shown in Figure 10. We have included the transitions T_6 , T_7 and T_8 in order to incorporate an equality condition to check the behavior of the algorithm. The transition-pair graph of the "shipping service" is shown in Figure 11. The grey switch states are included to instrument the loop. The arcs of the graph represent the transition-pairs generated through the state graph of Figure 10.

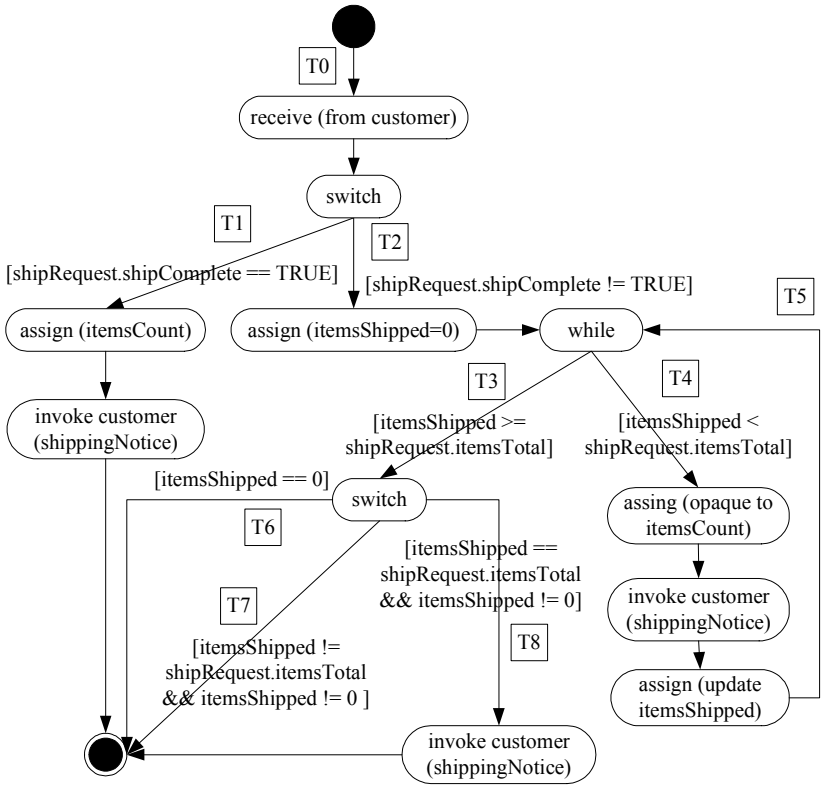

Figure 10. State graph of "shipping service" composition

The results obtained by TCSS-LS-for-BPEL are compared with those of a random generator. In all cases for our experiments, the stopping condition used for the generators was that of reaching 100% transition coverage or reaching 200000 generated test cases, the input variables of the compositions are integer and the input range uses 16 bits. For each

composition we carried out 100 runs with the generators, taking average values. All runs were carried out on a Pentium 4 processor 2.80GHz with a RAM memory of 512 MB.

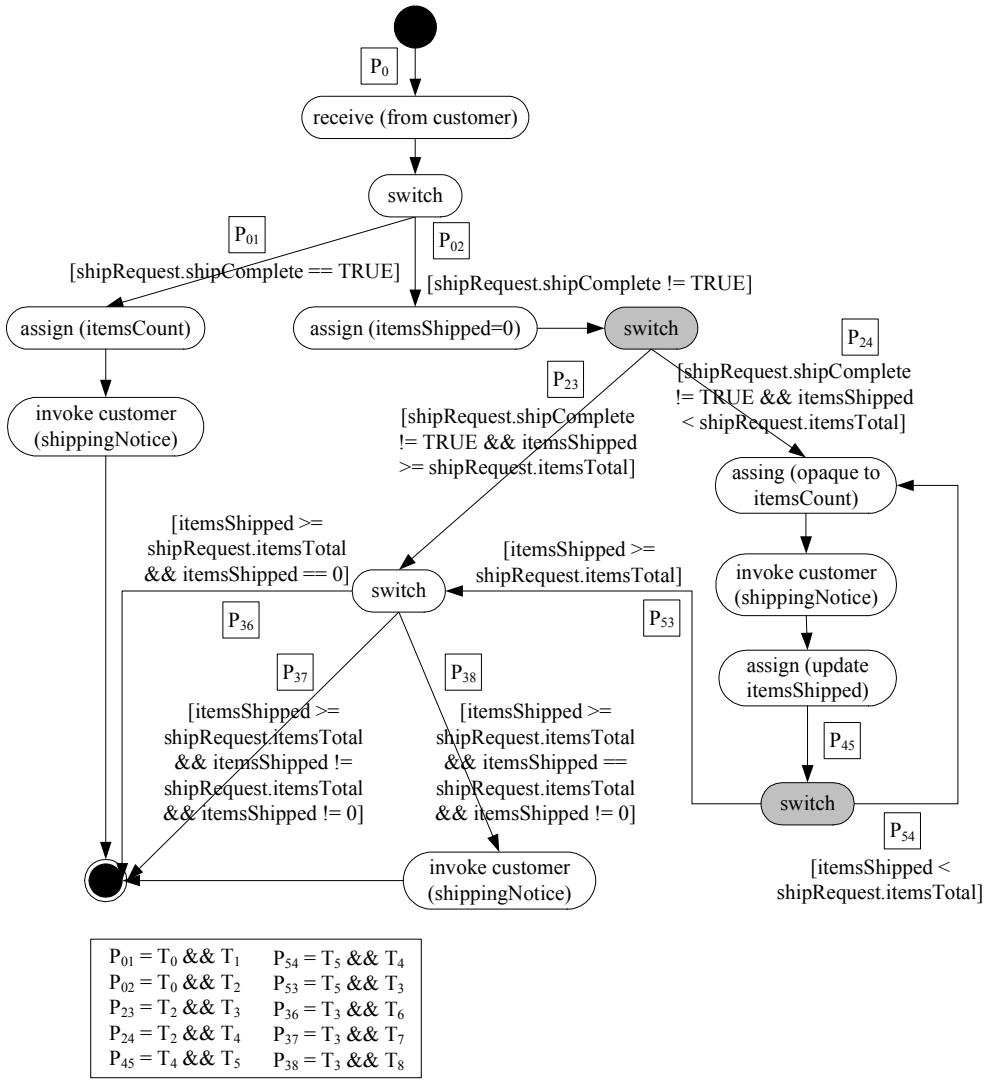

Figure 11. Transition-pair graph of "shipping service" composition

The results obtained by the two generators can be seen in Table 1. For both generators, the percentage of transition-pair coverage reached, the number of solutions that the generator creates to achieve this coverage and the time consumed (in seconds) are shown. TCSS-LS–for-BPEL generates few solutions and consumes less time than the random generator for both compositions. Moreover, the random generator does not achieve 100% coverage, whereas TCSS-LS-for-BPEL always reaches total coverage. On the other hand, the standard deviation of the number of solutions generated by TCSS-LS-for-BPEL has a small value for both compositions (29.3 for the "loan approval" composition and 43.1 for the "shipping service" composition).

Table 1. Results obtained for the compositions "loan approval" and "shipping service"

	Loan Approval			Shipping Service		
	% Coverage	Solutions	Time (s)	% Coverage	Solutions	Time (s)
TCSS-LS-for-BPEL	100	292	0,19	100	175	0,20
Random	99	54950	1,32	78	52990	1,01

During the search process, the test case generators create a large set of candidate solutions in order to cover all transition-pairs because some of these solutions reach transition-pairs that had already been covered by other ones. TCSS-LS-for-BPEL does not include all candidate solutions in the set S_{ij}, as according to the principle of the algorithm only the most diverse solutions are incorporated. To form the set of test cases that cover all transition-pairs we select some solutions from the sets S_{ij}. The selection process uses the set S_{ij} of the transition-pairs P_{ij} that reach the final state of the graph, as all transition-pairs are included in a path that finishes with some of these pairs. TCSS-LS-for-BPEL selects a solution from each of these sets that covers the greatest number of pairs. Table 2 shows an example of a set of test cases obtained from the sets S_{ij} for the "shipping service" composition. The first test case has been selected from set S_{01}, the second from set S_{36}, the third from set S_{37} and the fourth from set S_{38}.

Table 2. Example of test cases for the "shipping service" composition

Id	Input Variables			Transition-pairs Covered
	shipRequest.itemsTotal	Opaque values	shipRequest.shipComplete	
1	7033	-	TRUE	P_0, P_{01}
2	0	-	FALSE	P_0, P_{02}, P_{23}, P_{36}
3	28108	7381 5048 17839	FALSE	P_0, P_{02}, P_{24}, P_{45}, P_{54}, P_{53}, P_{37}
4	15387	9245 6142	FALSE	P_0, P_{02}, P_{24}, P_{45}, P_{54}, P_{53}, P_{38}

The left part of Figure 12 shows the coverage plots according to the number of solutions generated for the "shipping service" composition and the right part shows the coverage plots according to the time consumed. The horizontal axis represents the number of solutions generated (left part of the figure) or the time consumed (right part of the figure) to reach the accumulative percentage of transition-pair coverage represented by the vertical axis. As shown in these graphs, TCSS-LS-for-BPEL creates fewer solutions and consumes less time than the random generator to achieve each percentage of transition-pair coverage.

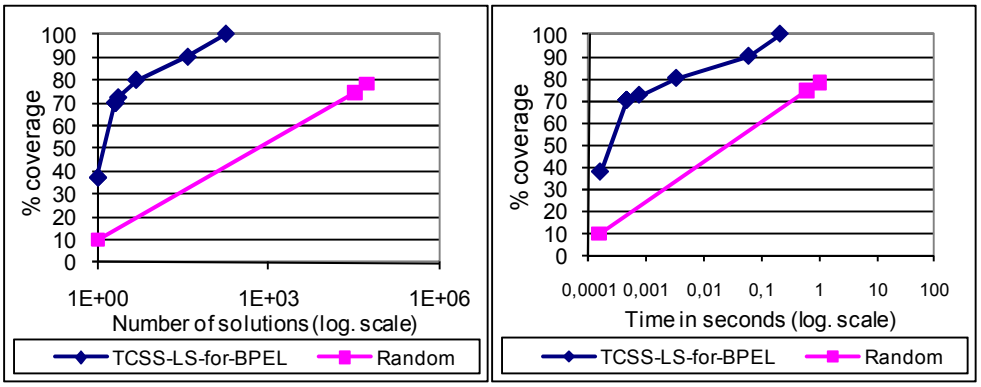

Figure 12. Coverage plots according to the number of solutions generated and the time consumed for the "shipping service" composition

6. Conclusions

This paper presents an approach based on the metaheuristic technique Scatter Search to automatically generate test cases for BPEL business processes and a group of transformations to represent the transition-pairs of the business processes. Both algorithm and transformations work together to generate test cases to fulfill a transition-pair coverage criterion, which allows us to test more different paths of execution of the business process than a transition coverage criterion, as each element of these paths is formed by two consecutive transitions. This approach, called TCSS-LS-for-BPEL, is an evolution of previous works.

The business process and its transition-pairs are represented as a transition-pair graph, where each arc corresponds to a transition-pair. TCSS-LS-for-BPEL handles a set S_{ij} in each arc of the graph, thus the general goal can be divided into several subgoals and each of them consists of generating solutions for a set S_{ij}. TCSS-LS-for-BPEL also provide procedures to work with solutions that have input variables with different and, in principle, an unknown number of values.

The results obtained show that TCSS-LS-for-BPEL can be applied to the test case generation of BPEL business processes and it outperforms the random generator. TCSS-LS-for-BPEL achieves 100% coverage in a short time and the evolution of the solutions generated quickly converges to the total coverage.

An immediate line of future work is the improvement of the algorithm to handle BPEL activities that enable the concurrent execution of other activities, such as flow. Further research is also needed to fully determine the scalability of our approach and, with this in mind, experimentation with real-world specifications is planned.

Acknowledgements

This work is supported by the Ministry of Science and Innovation (Spain) under the National Program for Research, Development and Innovation, projects Test4SOA (TIN2007-67843-C06-01) and Test4DBS (TIN2010-20057-C03-01).

References

[1] M.A. Ahmed, I. Hermadi, "GA-based multiple paths test data generator", Computers and Operations Research, 35(10), 2008, pp. 3107-3124.

[2] E. Alba, F. Chicano, "Observations in using parallel and sequential evolutionary algorithms for automatic software testing", Computers and Operations Research, 35(10), 2008, pp. 3161-3183.
[3] M. Alshraideh, L. Bottaci, "Search-based software test data generation for string data using program-specific search operators", Software Testing Verification and Reliability, 16(3), 2006, pp. 175-203.
[4] R. Blanco, J. García-Fanjul, J. Tuya, "A first approach to test case generation for BPEL compositions of web services using Scatter Search". In: Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops (2009) 131-140.
[5] R. Blanco, J. Tuya, E. Díaz, B. Adenso-Díaz, "A scatter search approach for automated branch coverage in software testing", Engineering Intelligent Systems, 15 (3), 2007, pp. 135-142.
[6] R. Blanco, J. Tuya, B. Adenso-Díaz, "Automated test data generation using a scatter search approach", Information and Software Technology, doi:10.1016/j.infsof.2008.11.001, 2008.
[7] P.M.S. Bueno, W.E. Wong, M. Jino, "Improving random test sets using the diversity oriented test data generation", in: Proceedings of the Second International Workshop on Random Testing, 2007, pp. 10-17.
[8] O. Bühler, J. Wegener, "Evolutionary functional testing", Computers and Operational Research, 35(10), 2008, pp. 3144-3160.
[9] C. Del Grosso, G. Antoniol, E. Merlo, P. Galinier, "Detecting buffer overflow via automatic test input data generation", Computers and Operational Research, 35(10), 2008, pp. 3125-3143.
[10] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, M. Bruno, "Search-based testing of service level agreements", in: Proceedings of the 9th conference on Genetic and Evolutionary Computation, 2007, pp. 1090-1097.
[11] E. Díaz, "Generación automática de pruebas estructurales de software mediante Búsqueda Tabú", PhD Thesis Department of Computer Science, University of Oviedo, 2004.
[12] E. Díaz, J. Tuya, R. Blanco, J.J. Dolado, "A tabu search algorithm for Software Testing", Computers and Operational Research, 35(10), 2008, pp. 3052-3072.
[13] W.L. Dong, H. Yu, Y.B. Zhang, "Testing BPEL-based Web Service Composition Using High-level Petri Nets". In: Proceedings of the 10th IEEE Int. EDOC Conf. Hong Kong (2006), pp. 441-444.
[14] J. García-Fanjul, J. Tuya, C. de la Riva, "Generating test cases specifications for BPEL compositions of web services using SPIN". In Proceedings of the Int. Workshop on Web Services – Modeling and Testing. Palermo (2006), pp. 83-94.
[15] M.R. Girgis, "Automatic test data generation for data flow testing using a genetic algorithm", Journal of Universal Computer Science, 11(6), 2005, pp. 898-915.
[16] F. Glover, M. Laguna, R. Martí, "Fundamentals of Scatter Search and Path Relinking", Control and Cybernetics 39(3), 2000, pp. 653-684.
[17] M. Harman, B.F. Jones, "Search-based software engineering", Information and Software Technology, 43(14), 2001, pp. 833-839.
[18] M. Laguna, R. Martí, Scatter Search: Methodology and Implementations in C, Kluwer Academic Publishers, Boston, MA, USA, 2002.
[19] Z. Li, M. Harman, R.M. Hierons, "Search algorithms for regression test case prioritization", IEEE Transactions on Software Engineering, 33(4), 2007, pp. 225-237.
[20] N. Mansour, M. Salame, "Data generation for path testing", Software Quality Journal, 12, 2004, pp. 121-136.
[21] T. Mantere, J.T. Alander, "Evolutionary software engineering, a review", Applied Soft Computing, 5(3), 2005, p. 315-331.
[22] P. McMinn, "Search-based software test data generation: a survey", Software Testing Verification and Reliability, 14(2), 2004, pp. 105-156.
[23] P. McMinn, M. Holcombe, "Evolutionary testing using an extended chaining approach", Evolutionary Computation, 14(1), 2006, pp. 41.64.
[24] L. Mei, W.K. Chan, T.H. Tse, "Data flow testing of service-oriented workflow applications", in: Proceedings of the 30th International Conference on Software Engineering (ICSE), Leipzig (Germany) 2008, pp 371-380.
[25] J. Miller, M. Reformat, H. Zhang, "Automatic test data generation using genetic algorithm and program dependence graphs", Information and Software Technology, 48, 2006, pp. 586-605.
[26] Organization for the Advancement of Structured Information Standards (OASIS), Web Services Business Process Execution Language (WSBPEL), URL: http://www.oasis-open.org.
[27] R. Sagarna, J.A. Lozano, "Scatter Search in software testing, comparison and collaboration with Estimation of Distribution Algorithms", European Journal of Operational Research, 169(2), 2006, pp. 392-412.
[28] S.R. Vergilio, A. Pozo, "A grammar-guided genetic programming framework configured for data mining and software testing", International Journal of Software Engineering and Knowledge Engineering, 16(2), 2006, pp. 245-267.
[29] H. Waeselynck, P. Thévenod-Fosse, O. Abdellatif-Kaddour, "Simulated annealing applied to test generation: landscape characterization and stopping criteria", Empirical Software Engineering, 12(1), 2007, pp. 35-63.
[30] A. Watkins, E.M. Hufnagel, D. Berndt, L. Johnson, "Using genetic algorithms and decision tree induction to classify software failures", International Journal of Software Engineering and Knowledge Engineering, 16(2), 2006, pp. 269-291.

[31] J. Wegener, A. Baresel, H. Sthamer, "Evolutionary test environment for automatic structural testing", Information and Software Technology, 43(14), 2001, pp. 841-854.
[32] M. Xiao, M. El-Attar, M. Reformat, J. Miller, "Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques", Empirical Software Engineering, 12(2), 2007, pp. 183-239.
[33] J. Yan, Z. Li, Y. Yuan, W. Sun, J. Zhang, "BPEL4WS unit testing: test case generation using a concurrent path analysis approach", in: Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE), Raleigh (USA) 2006, pp 75-84.