A first approach to test case generation for BPEL compositions of web services 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, which cost can be reduced by means of the use of techniques for automating this task. In this paper we present an approach based on the metaheuristic technique Scatter Search for the automatic test case generation of the BPEL business process. A transition coverage criterion is used as adequacy criterion.

1. Introduction

A challenging part of Software Testing entails the generation of test cases, which cost can be reduced by means of the use of techniques for automating this task. In service oriented architectures the deployment of software as a service has the objective, in the short or medium term, that these services will be invoked from other software or services. Thus, using well-established and automated testing techniques is essential to firstly assure the quality of the deployed services and, also, facilitate regression testing.

The search for an optimal solution in the test case generation problem has a great computational cost and for this reason these techniques 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 [19], and besides they obtain good results in test case generations [24]. In this paper we propose the use of the metaheuristic technique called Scatter Search in the automatic generation of test cases for web services compositions. The approach presented in the paper is an evolution of the TCSS-LS algorithm described in [7] which generates test cases for the branch coverage criterion for programs written in C.

The rest of the paper is organized as follows. The next section presents an overview of related studies. Section 3 details our Scatter Search approach for the automatic generation of test cases. In Section 4 we present the preliminary results and Section 5 presents the conclusions of this paper.

2. Background

2.1. BPEL business processes

BPEL specifications represent the behaviour 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 take 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. A structured activity that is not so common in other languages is the flow. Activities grouped in such

are concurrent, and so a flow completes when all of its activities have completed.

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 [30]. 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.

```
<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. Scatter Search technique

Scatter Search [18][21] 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 generate each time better solutions, according to quality and diversity criteria.

The basic scheme of the Scatter Search algorithm can be seen in Figure 2 [21]. 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 solution 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.

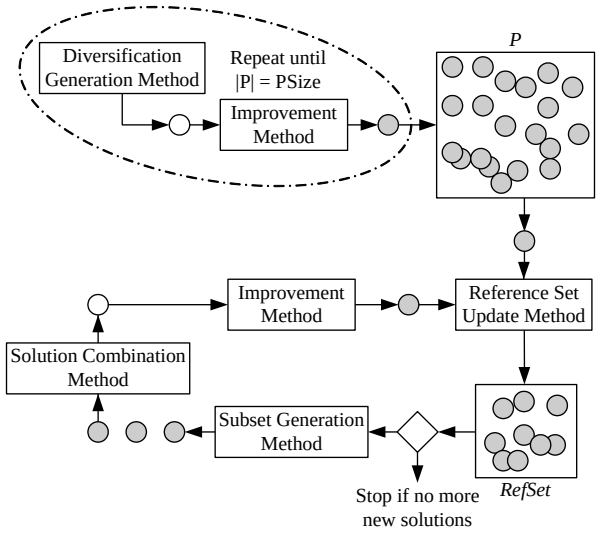

Figure 2. Basic scheme of Scatter Search

2.3. Metaheuristic search techniques in test case generation

The application of metaheuristic algorithms to solve problems in Software Engineering was proposed by the SEMINAL network (Software Engineering using Metaheuristic INnovative ALgorithms) and is widely explained in [10]. One of these applications is software testing, in which the testing problem is treated as a

search or optimization problem, as is shown in several surveys [24][26].

The most widely used metaheuristic technique in this yield is Genetic Algorithms. This technique is used in many papers to achieve several coverage criteria [28][35][3][36][2][17]. Other papers apply Genetic Algorithms to generate test cases to cover string predicates [4], to detect overflows [11], to regression test case prioritization [22], to train a series of decision tree in order to create rules for classifying test cases [34] and to generate test data that cause service level agreement violations in service-oriented systems [12].

Simulated annealing has been used to generate test cases to achieve several coverage criteria [23][36] and it has been used in the investigation of measures of landscape to apply this technique to test generation [33]. 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 [32]. Tabu Search has been used to obtain branch coverage [14] and path and loop coverage [13]. Simulated Repulsion has been used to generate diverse test data and evaluate the effect of diversity on data flow coverage and mutation testing [8]. Evolutionary algorithms are been used in the automation of functional testing [9], and their principles are been combined with an extended chaining approach to find test cases that cover a target [27]. Hill Climbing has been used in the regression test case prioritization [22]. Evolutionary Strategies has been used to achieve condition coverage [3]. Estimation of Distribution Algorithms has been used to obtain branch coverage [31].

Another metaheuristic technique that can be applied to automatic test case generation is Scatter Search. This technique has been used to solve many problems, as is shown in [25]. However, the only papers that use the Scatter Search technique to automate the generation of test cases are [6][7][31], which use this technique to obtain branch coverage.

2.4. Test case generation for BPEL business processes

There is not a great amount of research on the definition of testing methods for compositions of web services, and much of the published work has been devoted to monitoring approaches [1][5][29] Regarding the generation of test cases, Huang et al. [20] describe a method to test composite web services using model checking. The main differences with our aproach lie in the input and the testing criteria: they explicitly specify the behaviour of each web service in the composition (using OWL-S) and define the desired properties by hand. García-Fanjul et al. [16] do also use a model checker (SPIN) to generate test cases but they systematically select the test cases using a transition coverage criterion. Also, Dong et al [15] use High level Petri Nets to model BPEL business processes and generate test cases. Metaheuristic search techniques have not been used to generate test cases for BPEL business process yet.

3. Test case generation for BPEL business processes using scatter search

In this section we explain our adaptation of the Scatter Search technique to the automatic generation of test cases for BPEL business processes using a transition coverage criterion. In Section 3.1 we present the general aspects of our Scatter Search approach called TCSS-LS. Sections 3.2 and 3.3 show the search process of new test cases.

3.1. Problem approach

Our objective is to test BPEL compositions of web services. The input variables of the business process are the variables received from the web services (called partners in BPEL) that interact in the BPEL specification. A test case is defined by means of the values of the input variables and the transitions of the business process executed.

The BPEL specification does not directly include information about the behaviour of the different web services that participate in the business process, so a mock model will be constructed for each partner based upon its interface with the business process.

The BPEL business process is represented by means of a state graph where the nodes represent the states of the business process and the arcs represent the change of state from node i to node j when the associated arc decision is true, i.e., an arc represents a transition in the business process and it has its own Reference Set. By means of this state graph, it is possible to determine the transitions covered by the test cases generated, since the business process has been instrumented to know the followed path.

Figure 3 shows the state graph that represents the “loan approval” business process (a extract of the BPEL specification can be seen in Figure 1). Each transition of the business process is numbered as Tk. Note that some transitions are not numbered in the graph, as they are covered by test cases that cover other transitions or sets of transitions. For instance, if a test

suite covers transitions T2 and T4, then the transition from "invoke approver" to the final state will be covered.

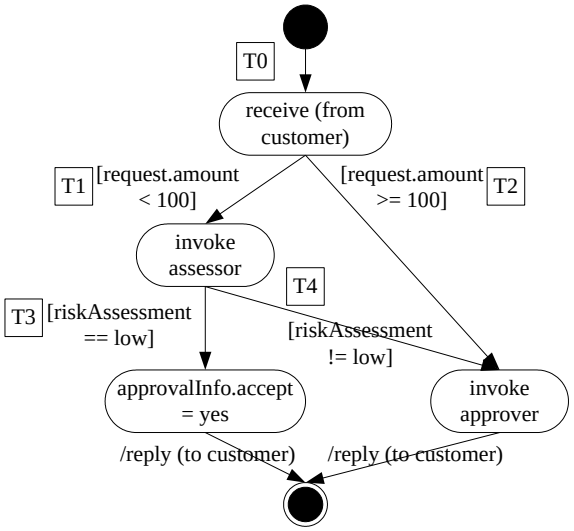

Figure 3. State graph of "loan approval" service

The goal of TCSS-LS is to generate test cases that allow all transitions of the business process to be covered. This general goal is divided in subgoals, each of which consists in finding test cases that reach a particular arc (transition) Tk of the state graph.

In order to reach the subgoals, the transitions of the state graph store information during the process of test case generation. This information allows the covered transitions to be known and is used to make progress in the search process. Each transition 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_k, where k is the number of the transition, and is formed by B_k elements $T_k^c = < \bar{x}_k^c, p_k^c, fb_k^c, fc_k^c >$, $c \in \{1..B_k\}$, where:

- $\bar{x}_k^c$ is a solution, i.e., a test case that reaches transition Tk. Each solution $\bar{x}_k^c$ 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 transitions of the previous transitions to transition Tk on the path that has been followed. Each input variable is represented as a vector since a web service can be invoked several times and each invocation provides an independent value.
- p_k^c is the path covered by the solution (test case), i.e., the sequence of the transitions of the state graph reached by the solution.
- fb_k^c is the distance to the sibling transition. This distance indicates how close the solution came to cover the sibling transition.
- fc_k^c is the distance to the next transition that has not been reached by the solution. This distance indicates how close the solution came to cover this transition.

An example of the state graph with the information stored in the Reference Sets of the transitions can be seen in Figure 4.

The procedure followed to calculate the maximum size B_k of the set of solutions of a transition Tk (S_k) and the calculation of the distances can be consulted in [7].

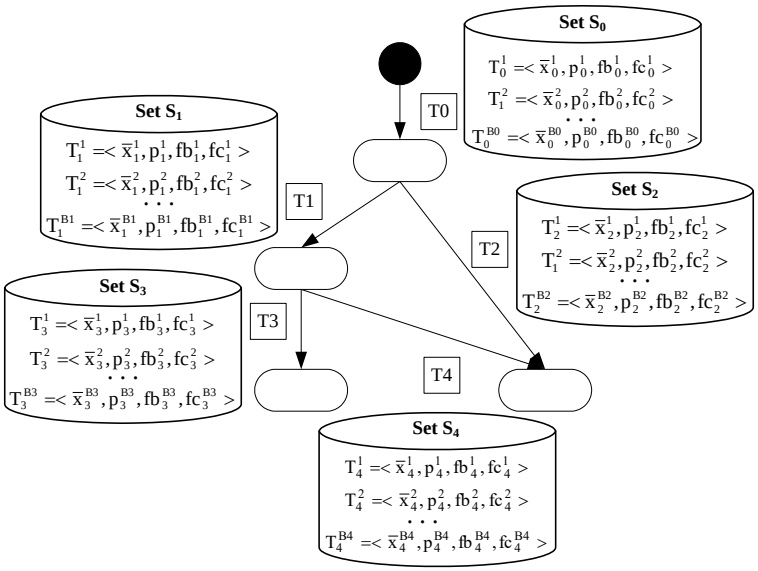

Figure 4. TCSS-LS state graph

TCSS-LS will try to make the sets as diverse as possible using a diversity function. Thus it tries to explore a wide search space in order to find solutions that can cover different transitions of the composition. The diversity of a solution of a set S_k is a measure related to the path covered by all solutions of the set.

3.2. Search process

The goal of TCSS-LS is to obtain maximum transition coverage, i.e., to find solutions that allow coverage of all the transitions of the state graph. As these solutions are stored in the transitions, our goal is therefore all the transitions to have at least one element in their set S_k. However, this goal cannot be reached when the composition under test has unfeasible transitions. Therefore, TCSS-LS also stops its execution when a maximum number of test cases has been generated. Initially, the sets S_k are empty and they are filled by the generator by means of its iterations.

Figure 5 shows the scheme of the TCSS-LS search process, which has been adapted to work with a state graph instead of a control flow graph. This search process starts generating random solutions that are stored in the Reference Set of transition T0 (S_0). The model of the composition is executed with each solution and the sets S_k of the transitions reached in this execution are updated. A first step in a run consists of configuring the partners with the values of the variables that they return to the business process when they are invoked. Then the iterations of the search process begin. In each iteration, TCSS-LS selects a transition (transition in evaluation) to form the subset of solutions from its Reference Set, which are used by the combination rules to generate the new solutions. These new solutions, which can be improved, are also executed in the model of the composition in order to update the Reference Sets S_k of the transitions reached, thus closing the cycle of execution. When a partner of the composition needs more values for an input variable of the business process under test, because it is invoked more times, the partner asks TCSS-LS for new values.

If the selected transition does not have at least two solutions to perform the combinations, a backtracking process, which is not considered in the original Scatter Search algorithm, is carried out. This backtracking process combines the Scatter Search technique with a local search method.

The backtracking process, the combination rules and the procedures followed to select a transition (similar to the selection of a node in the control flow graph), to form the subsets of solutions, to improve the solutions and to update the sets S_k using the diversity property can be consulted in [7].

The search process finishes when all transitions have been reached or the maximum number of test cases has been achieved.

The final solution of the generator consists of the test cases that cover the transitions of the state graph, which are in the sets S_k, the transition coverage reached and the time consumed in the search process. For example, a test case for the "loan approval" service defined in Figures 1 and 3 has the inputs request.amount = 50, risk.assessment = low and covers the path formed by the transitions T0, T1, T3.

3.3. Treatment of the unfixed number of values of an input variable

When a loop in the business process includes a web service invocation, the variable (or variables) returned by the partner has a different value in each iteration of the loop, which can cover different transitions of the business process and for this reason all these values of the variable must be recorded to generate the test cases. As the number of iterations of the loop is unknown in most cases, an input variable can take an unknown number of values in the execution of the web services composition and the vector $\overline{x}_j$ that represented it in the solution $\overline{x}_k^c$ can have different size in two specific solutions.

Thus the TCSS-LS algorithm described in [7] has been improved to include a new method to handle the unfixed number of values of an input variable.

When a solution is created randomly TCSS-LS generate a vector for each input variable situated inside a loop with a fixed number of values. Then the partners are configured with the vector of values of the variables that they handle and the business process is executed. Each partner consumes and returns a value of the vector of the variable in each invocation and when it is invoked and all the values of the variable are consumed it asks TCSS-LS for new values.

TCSS-LS searches these new values for the input variable among the solutions of the set S_k of the transition in evaluation Tk. It tries to find the values that are more diverse. Thus the "diversity of a variable" function is calculated over the subset $S_{k'}=\{T_{k'}^1,\ldots,T_{k'}^q\} \subseteq S_k$, $T_{k'}^c=<\overline{x}_{k'}^c;p_{k'};fb_{k'}^c;fc_{k'}^c>$, which represents the solutions stored in transition Tk that have not been used to give new values to the partner. The diversity value of a variable $\overline{x}_j$ is calculated according the "diversity of a variable" function defined as:

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

where index c=1..q covers the solutions of $S_{k'}$ and index z=1..r covers the values of the input variable $\bar{x}j$.

The less similar solution to the rest of solutions according to the values of variable $\bar{x}j$, i.e. the solution with the high value for *div_var()*, is selected and the values of the variable $\bar{x}j$ are given to the partner. Thus the size of the vector of values of a variable of a solution that is executed in the business process can be increased.

After the business process finishes its execution with a solution, it is analyzed to drop the values of the variables that have not been used, thus decreasing the size of the vectors.

On the other hand, the generation of new solutions and the diversity function presented in [7] has been adapted to handle the unfixed number of values of an input variable.

When TCSS-LS combines two solutions ($\bar{x}_k^c$, $\bar{x}_k^d$) g≠h in order to generate the new ones it applies the combination rules over each input variable as follow:

- $new1j = \begin{cases}\bar{x}j_k^{c_z} + \Delta_z & \text{if size of } \bar{x}j_k^c \geq z \\ \bar{x}j_k^{d_z} & \text{otherwise}\end{cases}$
- $new2j = \begin{cases}\bar{x}j_k^{c_z} - \Delta_z & \text{if size of } \bar{x}j_k^c \geq z \\ \bar{x}j_k^{d_z} & \text{otherwise}\end{cases}$
- $new3j = \begin{cases}\bar{x}j_k^{d_z} + \Delta_z & \text{if size of } \bar{x}j_k^d \geq z \\ \bar{x}j_k^{c_z} & \text{otherwise}\end{cases}$
- $new4j = \begin{cases}\bar{x}j_k^{d_z} - \Delta_z & \text{if size of } \bar{x}j_k^d \geq z \\ \bar{x}j_k^{c_z} & \text{otherwise}\end{cases}$

where index j covers all input variables, index z covers the positions of the vectors of the input variable $\bar{x}j$ and Δ_z is defined as:

$$\Delta_z = \begin{cases}\dfrac{\left|\bar{x}j_k^{c_z} - \bar{x}j_k^{d_z}\right|}{2} & \text{if size of } \bar{x}j_k^c \geq z \text{ and size of } \bar{x}j_k^d \geq z \\ 0 & \text{otherwise}\end{cases}$$

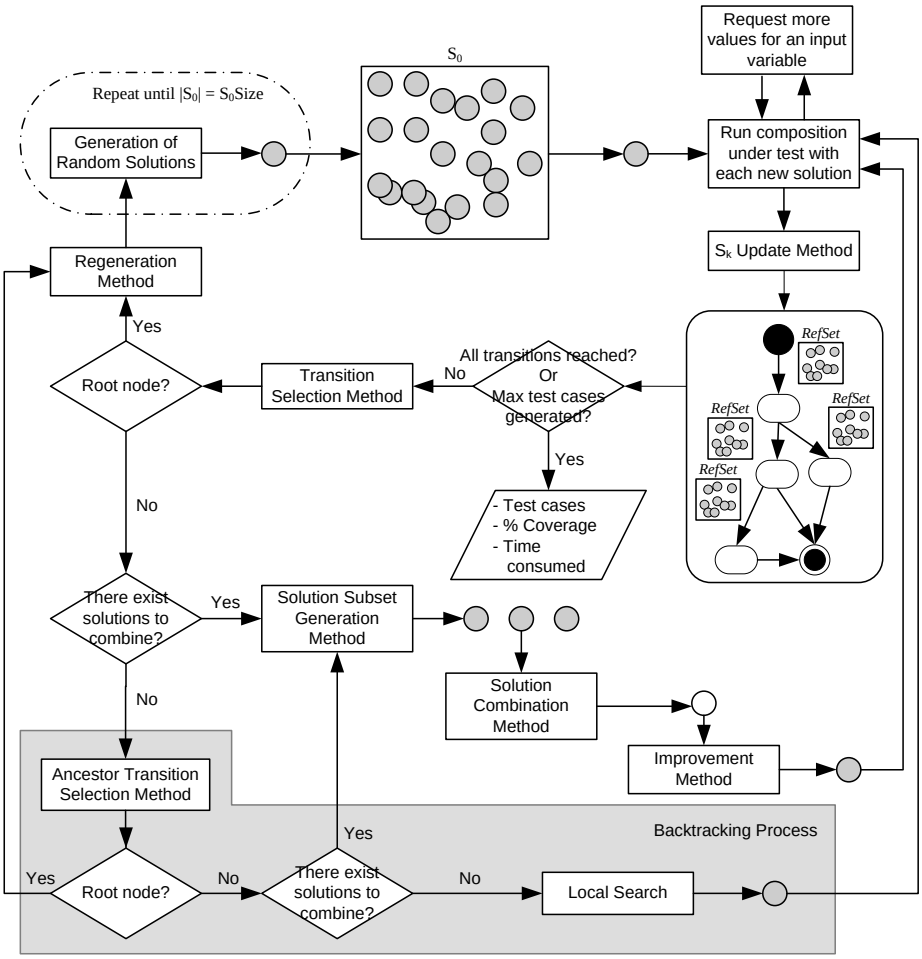

Figure 5. TCSS-LS scheme (improved)

The solutions generated by means of the combinations rules are executed in the composition under test and whether a partner consumes all values of the vector of a variable, it asks TCSS-LS for new values as is indicated above.

When TCSS-LS uses the local search method, each new solution is generated by means of the modification of all positions of the vector of a specific variable, according to the procedures described [7].

TCSS-LS applies the diversity function when it needs to remove the less diverse solution from a set S_k in the updating process. The diversity function is applied over the subset $S_{p^*}=\{T_{p^*}^1,\ldots,T_{p^*}^q\} \subseteq S_k$, $T_{p^*}^c=<\bar{x}_{p^*}^c;p_{p^*};fb_{p^*}^c;fc_{p^*}^c>$, which represents the solutions stored in transition Tk that cover the path p_{p^*} with more occurrences in the set S_k. The diversity value of a solution is calculated according the diversity function defined as:

$$div(<\bar{x}j_{p^*}^m;S_{p^*}>;S_{p^*}) = \sum_{c=1..q}\sum_{j=1..n}\sum_{z=1..r}\begin{cases}\dfrac{\left|\bar{x}j_{p^*}^{m_z}\right|}{range_j} & \text{if size of } \bar{x}j_{p^*}^c < z \\ \dfrac{\left|\bar{x}j_{p^*}^{c_z}\right|}{range_j} & \text{if size of } \bar{x}j_{p^*}^m < z \\ \dfrac{\left|\bar{x}j_{p^*}^{m_z} - \bar{x}j_{p^*}^{c_z}\right|}{range_j} & otherwise\end{cases}$$

where index c=1..q covers the solutions of S_{p^*}, index j=1..n covers the input variables, index z=1..r covers the values of the input variable $\bar{x}j$ and $range_j$ is the range of values of the input variable $\bar{x}j$.

4. Case studies

We applied our first approach to two BPEL specifications: the "loan approval" (which has also been used to exemplify the method, see Figure 1 and Figure 3) and the "shipping service". Both of these specifications were originally published within the standard BPEL4WS and have been extensively referenced in the literature on web services testing. 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 is shown in Figure 6. We have included the transitions T5 and T6 in order to increase the complexity of the composition with an equality condition.

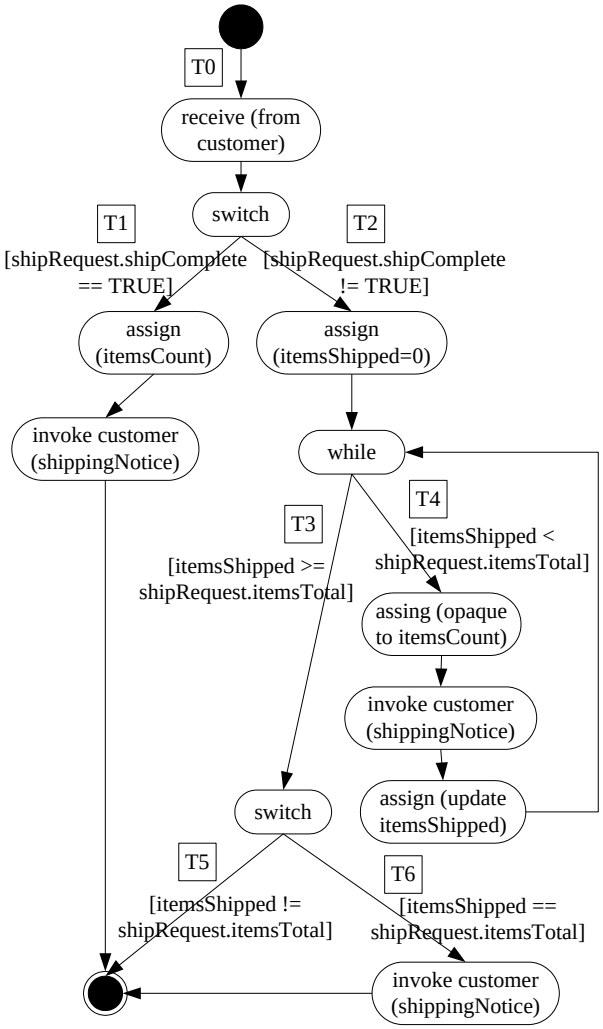

Figure 6. State graph of "shipping service" composition

The results obtained by TCSS-LS 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.

Table 1 shows the results obtained for both generators: percentage of transition coverage reached, the number of test cases that the generator creates to achieve this coverage and the time consumed (in seconds). Note that the test case generators use a large set of test cases to cover all transitions, because during the search process they generate test cases that reach transitions that had already been covered by other test

cases. However not all of them are included in the sets S_k, (only the most diverse test cases are incorporated). On the other hand, not all of the test cases stored in the sets S_k form the set of test cases used in the test process of the business process. To obtain the minimum set of test cases that are executed in the composition under test to cover all transitions during the test process we select a test case from each set S_k. TCSS-LS generates few test cases and consumes less time than the random generator for both compositions. Besides the random generator does not achieve 100% coverage, whereas TCSS-LS always reaches total coverage.

Table 1. Results from "loan approval" and "shipping service" compositions

	Loan Approval			Shipping Service		
	% Cov.	Test Cases	Time (s)	% Cov.	Test Cases	Time (s)
TCSS-LS	100	290	0.19	100	144	0.23
Random	99	54940	1.31	75	36436	0.69

Figure 7 despites the evolution of the number of test cases generated for both generators for "shipping service" composition. The horizontal axis represents the number of test cases generated to achieve the accumulated percentage of transition coverage represented in the vertical axis. Figure 8 shows the number of seconds consumed by the generators for the "shipping service" composition to achieve each accumulated percentage of transition coverage. In these figures, it can be observed that TCSS-LS generates few test cases and consumes less time than the random generator to reach each percentage of transition coverage.

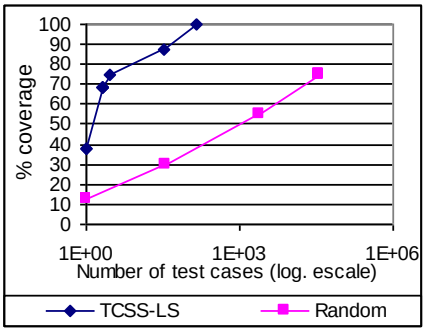

Figure 7. Evolution of the test cases generated for the "shipping service" composition

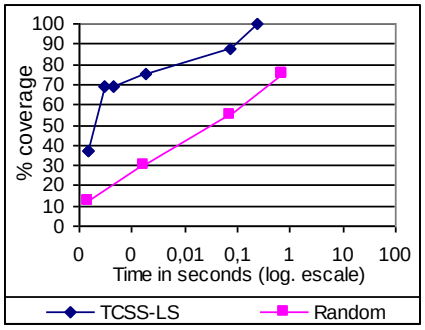

Figure 8. Evolution of time for the "shipping service" composition

An example of a set of test cases obtained from the sets S_k for the "shipping service" composition is shown in Table 2. The selection process starts in the sets S_k of the transitions that end in the final state and finishes when all transitions have been covered by the test cases selected. The first test case is selected from the set S_k of transition T6 and covers transitions T0, T2, T4, T3 and T6. The second test case is selected from the set S_k of transition T5, which covers the transition T5 that has not been covered by the first test case. The third test cases is selected from the set S_k of transition T1 and now all transitions of the "shipping service composition" have been covered by the test cases selected.

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

Id	Input Variables			Transitions Covered
	shipRequest. itemsTotal	opaque values	shipRequest. shipComplete	
1	31767	1153 13691 16923	FALSE	T0, T2, T4, T3, T6
2	6435	32280	FALSE	T0, T2, T4, T3, T5
3	10247	-	TRUE	T0, T1

5. Conclusions and future work

This paper presents our first approach based on the metaheuristic technique Scatter Search for the automatic test case generation of the BPEL business process to fulfil a transition coverage criterion. This approach, called TCSS-LS, is an evolution of a previous work.

The business process is modelled and represented by a state graph. TCSS-LS handles a set of solutions in each transition of the graph, thus facilitating the division of the general goal in subgoals, and provides

mechanisms to handle the unfixed number of values of the input variables.

The results obtained show that TCSS-LS can be applied to the test case generation of BPEL business processes and it outperforms the random generator.

Lines of future works are the use of other adequacy criteria as transition-pair coverage criterion, the improvement of TCSS-LS to handle the concurrent execution of activities in BPEL web services compositions, and the experimentation with real-life specifications.

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 RePRIS (TIN2007-30391-E).

References

[1] W.M.P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinnat, E. Verbeek, "Conformance Checking of Service Behavior". *ACM Transactions on Internet Technology*, 8(3), 2008

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

[3] 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.

[4] 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.

[5] L. Baresi, S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes", *in: Proceedings of the 3rd International Conference on Service Oriented Computing*, Amsterdam, 2005, pp. 269-282.

[6] 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.

[7] 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.

[8] 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.

[9] O. Bühler, J. Wegener, "Evolutionary functional testing", *Computers and Operational Research*, 35(10), 2008, pp. 3144-3160.

[10] J. Clarke, J.J. Dolado, M. Harman, R.M Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, M. Shepperd, "Reformulating software engineering as a search problem", *IEE Proceedings – Software*, 150(3), 2003, pp. 161-175.

[11] 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.

[12] 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.

[13] E. Díaz, "Generación automática de pruebas estructurales de software mediante Búsqueda Tabú", PhD Thesis Department of Computer Science, University of Oviedo, 2004.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] F. Glover, M. Laguna, R. Martí, "Fundamentals of Scatter Search and Path Relinking", *Control and Cybernetics* 39(3), 2000, pp. 653-684.

[19] M. Harman, B.F. Jones, "Search-based software engineering", *Information and Software Technology*, 43(14), 2001, pp. 833-839.

[20] H. Huang, W-T Tsai, R. Paul and Y. Chen, "Automated Model Checking and Testing for Composite Web Services", in: *Proceedings of the Eighth IEEE International Symposium*

on Object-Oriented Real-Time Distributed Computing, Seattle (USA) 2005, pp 300-307.
[21] M. Laguna, R. Martí, *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers, Boston, MA, USA, 2002.
[22] 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.

[23] N. Mansour, M. Salame, "Data generation for path testing", *Software Quality Journal*, 12, 2004, pp. 121-136.

[24] T. Mantere, J.T. Alander, "Evolutionary software engineering, a review", *Applied Soft Computing*, 5(3), 2005, p. 315-331.

[25] R. Martí, "Scatter Search—Wellsprings and Challenges", *European Journal of Operational Research*, 169(2), 2006, pp. 351–358.

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

[27] P. McMinn, M. Holcombe, "Evolutionary testing using an extended chaining approach", *Evolutionary Computation*, 14(1), 2006, pp. 41.64.

[28] 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.

[29] O. Moser, F. Rosenberg, S. Dustdar, "Non-Intrusive Monitoring and Adaptation for WSBPEL". In: *Proceedings of the 17th International World Wide Web Conference*. Beijing (2008), pp. 21-25.

[30] Organization for the Advancement of Structured Information Standards (OASIS), Web Services Business Process Execution Language (WSBPEL), URL: `http://www.oasis-open.org`.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.