CIT. Journal of Computing and Information Technology, Vol. 24, No. 3, September 2016, 271–282
doi: 10.20532/cit.2016.1002727

# Generating an Automated Test Suite by Variable Strength Combinatorial Testing for Web Services

Yin Li, Zhi-an Sun and Jian-Yong Fang

Jiangsu Institute of Automation, Lianyungang, China

Testing Web Services has become the spotlight of software engineering as an important means to assure the quality of Web application. Due to lacking of graphic interface and source code, Web services need an automated testing method, which is an important part in efficiently designing and generating test suite. However, the existing testing methods may lead to the redundancy of test suite and the decrease of fault-detecting ability since it cannot handle scenarios where the strengths of the different interactions are not uniform. With the purpose of solving this problem, firstly the formal tree model based on WSDL is constructed and the actual interaction relationship of each node is made sufficient consideration into, then the combinatorial testing is proposed to generate variable strength combinatorial test suite based on One-test-at-a-time strategy. At last test cases are minimized according to constraint rules. The results show that compared with conventional random testing, the proposed approach can detect more errors with the same amount of test cases which turning out to be more ideal than existing ones in size.

*ACM CCS (2012) Classification:* Software and its engineering → Software creation and management → Software verification and validation → Software defect analysis → Software testing and debugging

*Keywords*: testing Web Services, automated testing, combinatorial testing, WSDL, constraint rules

## 1. Introduction

As a type of implementation technologies for Service-Oriented Architecture (SOA), Web Services (WS) have provided an interoperability distributed application platform using the standard Web protocol, to implement the features such as open standard-based platform, loosely coupled platform and cross-platform. With the popularity of this technology, the requirement of quality and correctness is more and more critical, however, guaranteeing the Web Services software quality and reliability has become a tough issue in software engineering field.

Software testing is an important means of ensuring software quality. However, in order to guarantee the quality of Web Services, it must be tested in detail. Because of the complexity of Web Services technology specification, the variability of running state, if different services are tested one by one with similar functions, this work would be very repetitive and easily lead to human errors. Meanwhile, Web Services are invoked via service interface and not equipped with a visual graphic user interface which makes manual testing more difficult. Therefore, traditional test cases approach by handwork will not meet the requirement of testing, in order to test it in detail, the automatic approach is needed.

Regardless of stateless and stateful service, the testing single operation of Web Services is necessary. The tester can derive black box testing suite from data description information based on standard XML specification related files. Up to now, the researchers have presented a few approaches discussing automatic generating test cases based on Web Services Description Language (WSDL) specification file, however, testing the single operation of Web Service still faces some challenges, such as high redundancy test cases, lack of pertinence after reduction of use cases and poor fault detection ability. It is hard to achieve the goal of improving the test efficiency and Web Services quality.

Therefore this study aims to propose an automatic test data generating approach based on combinatorial testing and data constraints rules in order to obtain optimal test cases for a single operation of Web Service. As Figure 1 shows, in the overall approach, firstly, a formal type model for XML schema is constructed by input/output elements of the single operation in WSDL file. Secondly, test suite is obtained on the basis of the model built by previous step, then the test data is optimized by the means of combinatorial testing, and finally the statements or paths for Web Services could be covered by the optimal test cases.

This paper is organized as follows. In addition to Section 1 mentioned above, Section 2 presents the background and related works in the area of testing Web Services automatically. Section 3 demonstrates models and their definitions. Section 4 studies the generation of test data based on the model. Section 5 shows how to obtain the optimal test cases. An application example and conclusion are presented in Section 6 and Section 7 separately.

## 2. Background and Related Works

Service-Oriented Architecture (SOA) is not a new concept which has got wide attentions and applications in the recent years. As one of the implemented technologies for this architecture, automatic test for Web Services has become an important research in software testing area [1], [2]. Some researchers divide Web Services into stateless and stateful services and then separate them into three levels, namely, testing the single operation of Web Services, the operations sequence of Web Services and composite Web Services [2]-[5].

As the development of Web Services testing, some approaches have been proposed to generate test cases for single operation of Web Services through WSDL-based files. Tsai et al. [6] showed an approach by extending WSDL in order to support testing Web Services that incorporates with sequence specifications as well as input-output dependencies. Bai et al. [7] proposed an approach to generate test cases for Web Services from WSDL, which consisted

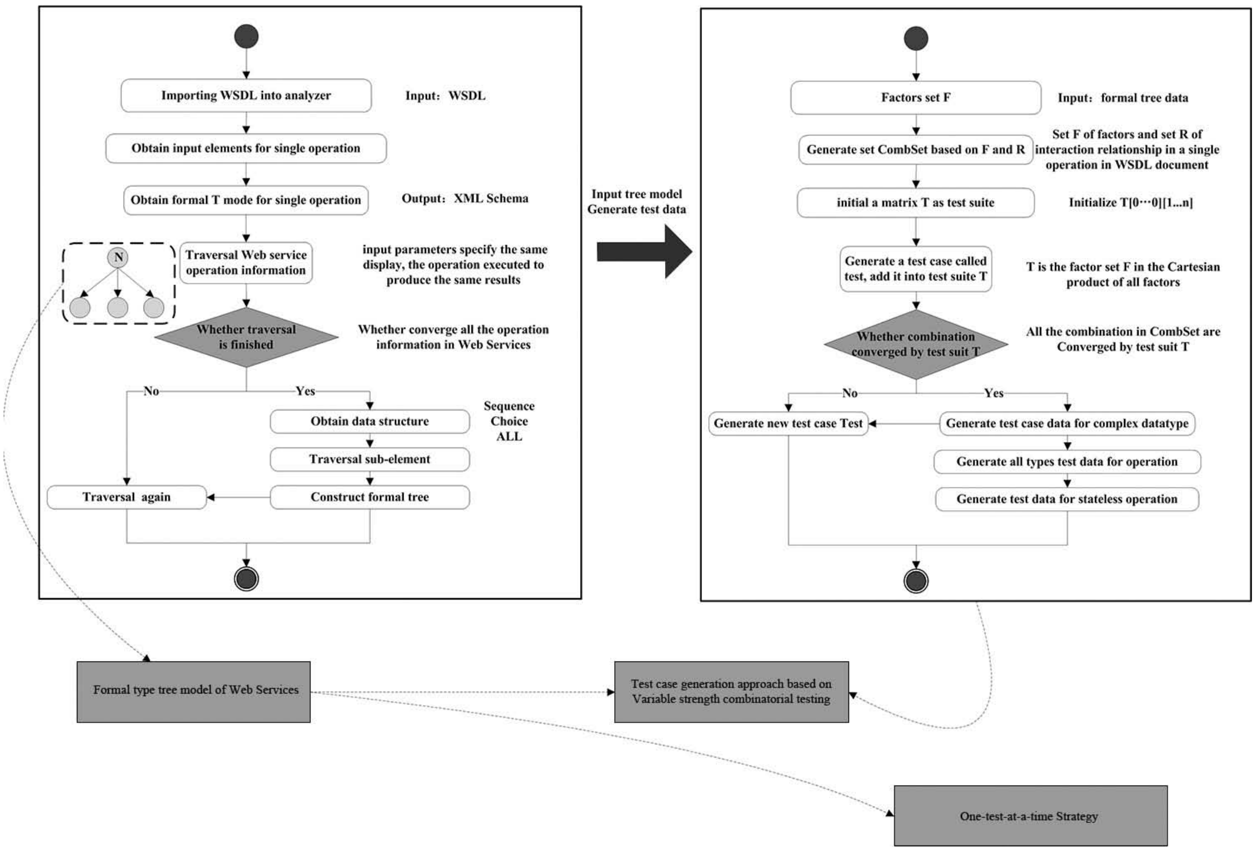
Importing WSDL into analyzer
Input: WSDL
Obtain input elements for single operation
Obtain formal T mode for single operation
Output: XML Schema
N
Traversal Web service operation information
input parameters specify the same display, the operation executed to produce the same results
Whether traversal is finished
Whether converge all the operation information in Web Services
No
Yes
Obtain data structure
Sequence Choice ALL
Traversal sub-element
Traversal again
Construct formal tree
Input tree model Generate test data
Factors set F
Input: formal tree data
Generate set CombSet based on F and R
Set F of factors and set R of interaction relationship in a single operation in WSDL document
initial a matrix T as test suite
Initialize T[0…0][1...n]
Generate a test case called test, add it into test suite T
T is the factor set F in the Cartesian product of all factors
Whether combination converged by test suit T
All the combination in CombSet are Converged by test suit T
No
Yes
Generate new test case Test
Generate test case data for complex datatype
Generate all types test data for operation
Generate test data for stateless operation
Formal type tree model of Web Services
Test case generation approach based on Variable strength combinatorial testing
One-test-at-a-time Strategy

*Figure 1.* The workflow of generating test data for individual operation.

of test data generation, test operation generation, operation flow generation and test specification. [8] firstly proposed the concept to construct abstract model through WSDL document. Subsequently, [9] presented the formal type tree on the basis of his work, they considered the data structure of complex data types, combined the sub-elements in complex data types and finally generated test data based on this model. Due to lack of consideration on test cases reduction in generation phase, test data tends to result in a combinatorial explosion by Cartesian method, it is not conductive to carry out large-scale testing. Moreover, some researchers proposed mutation testing technology [10]-[13], Xu and Offutt used data perturbation based on XML data type to generate test data for Web services. Siblini and Mansour [12] realized the testing for Web Services by document mutation. In China, Jiang [13] proposed the generation method of Web Services testing data based on contract mutation which can automatically generate initial random test data via using WSDL documents and select test data on the basis of contract mutation. While mutation testing needs a lot of computer resource, and the quality of test cases depends on the merits of the mutation operator, it is not a regular means of Web Services testing.

In combinatorial testing area, after analyzing the fault-reporting record of Moliza browser, Kuhn and Reilly [14] found there were more than 70% of the errors caused by the interaction of the two parameters and over 90% of the errors caused by the interaction of three parameters. Furthermore, Kuhn and Wallace [15] studied the availability of combinatorial testing applied into large-scale distributed systems and found that failures triggered in such system were generally cased by 4-6 interaction parameters at most. Schroeder [16] demonstrated that the error detection ability of N-dimensional combinatorial testing is much higher than the same size of random testing. Currently, combinatorial testing is widely used in compatibility testing, GUI testing and Web application testing [17], [18].

Focusing on the studies mentioned above, although there were a few test cases generation methods, some limitations still exist in this research area. On the basis of paper [6]-[9], our study hereby constructs a formal model of the type tree by analyzing the WSDL document and taking full account of the actual interactions between factors. From a new perspective of testing, this paper provides a concept of test suite reduction combing variable strength combinatorial testing and data constraints model, which ultimately could reduce test cost and improve test efficiency.

## 3. Model and Definition

### 3.1. WSDL Language

Web Services Description Language (WSDL) is an XML-based language which used to describe the location of the service, operations, methods and service information involved in it. Tsai [6] presented an extension to WSDL, added the constraint conditions, involved constraint information and semantic information. The extended WSDL files include built-in data types, simple data types and complex data types. Simple data types are defined with the element <xs:simpleType> which has representation constraints with built-in data types. Complex data types are defined with the element <xs:complexType> which has integrity constraints with the sub-element. There are three indication relationships among the sub-elements, such as *Sequence, Choice and All*. *Sequence* means that sub-elements may appear in the same order as they are defined in the schema file. *Choice* means that only one of sub-elements must appear (similar as enumeration). *All* means that sub-elements may appear in any order while the constraining facets *maxOccurs* cannot be greater than 1 [19].

According to the relationships between pairs of nodes and between nodes and data types, XML Schema classifies constraints as integrity constraints and representation constraints, these constraints can be classified as boundary constraints and non-boundary constraints as well, the detailed classification of these elements is shown in Table 1.

*Table 1*. Data Constraints Relationship.

| Relationship of the Constraints | Classification of the Constraints | Constraints Facets |
|---|---|---|
| **Constraints Between Nodes** | Integrity Constraints | *unique, maxOccurs, minOccurs, niliable, use, length, maxExclusive, maxInclusive* |
| | Representation Constraints | *maxLength, minExclusive, minInclusive, minLength, minLength, totalDigits, fractionDigits, pattern, whitespace, enumeration* |
| **Constraints Between Boundary** | Boundary Constraints | *maxOccurs, minOccurs, length, maxExclsive, maxInclusive, maxLength, minExclsive, minInclusive, minLength, totalDigits,* |
| | Non-boundary Constraints | *enumeration, use, fractionDigits, pattern, nilable, whitespace, unique* |

## 3.2. A Formal Model of Type Tree

This paper focuses on the test data generation for a single operation of Web Service and then defines the formal model of type tree based on the model proposed in Ma et al. [9] research as shown in Figure 2. However, the model can describe all the data input elements in the single operation of Web Service completely.

A formal model of type tree can be defined as follows:

**Definition 1.** An input element type model of the operation can be modeled as a formal tree set $T$ ($N$, $S$, $B$, $n_r$, $IC$, $RC$, $EE$, $ED$), where:

$N$ is a finite set of all the sub-elements in the complex data types; $S$ is a finite set of simple data types nodes in the input element type definition; $B$ is a set of built-in data type nodes in input element type definition; $n_r$ is a set of root nodes; $IC$ is a finite set of constraints between complex data type and its root nodes and between complex data type and its sub-elements,

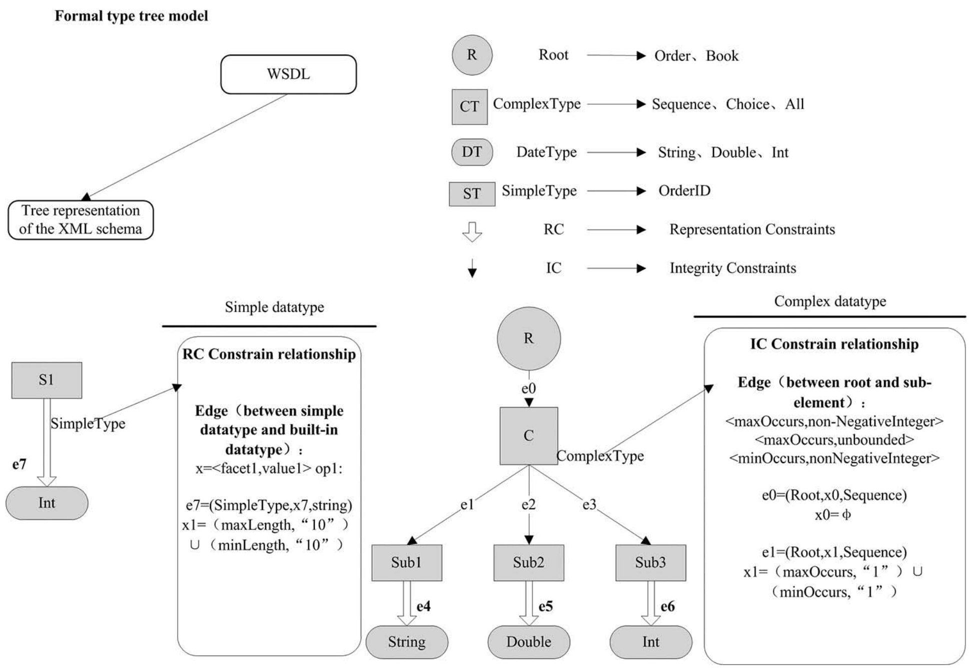
Formal type tree model
WSDL
Tree representation of the XML schema
R Root → Order、Book
CT ComplexType → Sequence、Choice、All
DT DateType → String、Double、Int
ST SimpleType → OrderID
RC → Representation Constraints
IC → Integrity Constraints
Simple datatype
S1
SimpleType
e7
Int
RC Constrain relationship
Edge（between simple datatype and built-in datatype）：
x=<facet1,value1> op1:
e7=(SimpleType,x7,string)
x1=（maxLength, "10"）∪（minLength, "10"）
R
e0
C
ComplexType
e1
e2
e3
Sub1
Sub2
Sub3
e4
e5
e6
String
Double
Int
Complex datatype
IC Constrain relationship
Edge（between root and sub-element）：
<maxOccurs,non-NegativeInteger>
<maxOccurs,unbounded>
<minOccurs,nonNegativeInteger>
e0=(Root,x0,Sequence)
x0=φ
e1=(Root,x1,Sequence)
x1=（maxOccurs, "1"）∪（minOccurs, "1"）

*Figure 2*. Formal model of type tree.

namely integrity constraints defined in WSDL file. *RC* is a finite set of facet constraints between simple data type and built-in data type, namely representation constraints defined in WSDL file; *EE* is a finite set of edges: $\forall\ e \in EE$, denoted as $e\ (p, x, c), p \in N \cup n_r, c \in N, x \in IC \cup \{\varnothing\}$; *ED* is a finite set of edges: $\forall\ e \in ED$, denoted as $e\ (p, x, c), p \in N \cup S, c \in B, x \in RC \cup \{\varnothing\}$.

According to the definition of model *T* in Definition 1, the construct algorithm is presented as follows:

## 3.3. Variable Strength Combinatorial Model

According to the papers [20]-[22], combinatorial testing model is defined as follows:

Let the software under testing (SUT) have $n$ parameters, and each factor $f_i$ has $a_i$ $(1 \le i \le n)$ discrete values. Let $F = \{f_1, f_2, ..., f_n\}$ denote the set of factors, and $V_i = \{1, 2, ..., a_i\}$ $(1 \le i \le n)$ denote the value set of factor $f_i$.

**Definition 2.** Combinatorial testing test case set

Let $n$-tuple

$test = (v_1, v_2, ..., v_n)\ (v_1 \in V_1, v_2 \in V_2, ..., v_n \in V_n)$

*Algorithm 1.* Obtaining formal model of type tree *T.*

```
Input: WSDL document
Output: A type tree formal model of a single operation in WSDL:
        $T\,(N, S, B, n_r, IC, RC, EE, ED)$
Initial:
$N = \varnothing, S = \varnothing, B = \varnothing, n_r = \varnothing, IC = \varnothing, RC = \varnothing, EE = \varnothing, ED = \varnothing$
    // Initialize all data type sets in T model $n_r \cup \{n\}$;
    // Get new data type node $n$ by analyzing the WSDL document
Generate_T($n$); // Generate data type model of Node $n$
Generate_T($n$): // The algorithm of construct data type model of node $n$
Switch ($n$)
Case Complex data type: // if $n$ is Complex data type
$N = N \cup \{m\}, EE = EE \cup \{e\}, n = m$;
    // $m$ is one of the key words, such as sequence, choice, all, generate edge
      $e\ (n, x, m), x = \varnothing$, take new node $m$ as parent node
For each SubElement in $n$
    $N = N \cup \{m\}, EE = EE \cup \{e\}, IC = IC \cup \{x\}, n = m$;
      // Generate edge $e\ (n, x, m)$ and restriction facet set $IC$, take new node $m$ as parent node
    Generate_T(SubElement) // iteration traversal every sub-element
    Case Simple data type: // $n$ is simple data type
    If ($n$ has restriction constraints)
      // whether simple data type $n$ contains restriction constraints
    {
        $S = S \cup \{n\}, ED = ED \cup \{e\}, RC = RC \cup \{x\}, n = m$;
          // $m$ is the next node of $n$, and the built-in data type. This algorithm generates edge
            $e\ (n, x, m)$ and user defined restriction constraints set $RC$ as well as take new
            node $m$ as parent node
    }
    else
    {
      $B = B \cup \{m\}, ED = ED \cup \{e\}, RC = RC \cup \{x\}$;
        // $m$ is the next node of $n$, is built-in data type, algorithm generates edge $e\ (n, x, m)$
          and default restriction constraints set $RC$
    }
    Case built-in data type: // $n$ is built-in data type
    $B = B \cup \{m\}, ED = ED \cup \{e\}, RC = RC \cup \{x\}$;
     // $m$ is the next node of $n$, and is built-in data type, algorithm generates edge $e\ (n, x, m)$
       and default restriction constraints set $RC$
End for;
Output the T model of single operation
```

call a test case for SUT, and the set consists of these test cases called test case set for SUT.

**Definition 3.** *N* dimensional combinatorial coverage

Given $A = (a_{i,j})_{m\times n}$ is $m \times n$ array, where $j$th column denotes the factor $f_i$ of SUT and all elements of this column come from the finite set $V_i = \{1, 2, ..., n\}$, that is $a_{i,j} \in V_j$. If every $m \cdot n$ $(2 \le N \le n)$ sub-arrays contain all value combinations of such $N$ columns (or factors), then $A$ is an $N$-way fixed strength covering array or a fixed strength covering array with strength $N$, and it could be denoted as $CA$ $(m : N : F)$.

**Definition 4.** Interaction relation set

$A$ subset $r_k \in R$ $(k = 1, 2, ..., t)$ could be named as interaction coverage requirement, or coverage requirement for short. And the collection $R$ could be named as the interaction relationship of SUT.

Given $A = (a_{i,j})_{m\times n}$ is $m \times n$ array, where $j$th column denotes the factor $f_i$ of SUT and all elements of this column come from the finite set $V_i = \{1, 2, ..., n\}$, that is $a_{i,j} \in V_j$. For a coverage requirement $r_k \in R$, if the sub-array consists of all factors, then $A$ satisfies $r_k$, if $A$ satisfies all coverage requirements in an interaction relationship $R$, then $A$ is a variable strength covering array for $R$ and it could be denoted as $VCA$ $(m, F, R)$.

Accordingly, the variable strength covering array for $R$ should cover all combinations in the set:

$$CombSet = \bigcup_{k=1}^{t} CombSet_k$$

where the $CombSet_k$ $(k = 1, 2, ..., t)$ covers the coverage requirement $r_k$:

$$CombSet_k = \{(v_{k,1}, v_{k,2}, ..., v_{k,nk})| \\ v_{k,1} \in V_{k,1}, v_{k,2} \in V_{k,2}, ..., v_{k,nk} \in V_{k,nk}\}.$$

## 3.4. Data Constraints Model

This paper improves the constraints model which was proposed by Hou et al. [23]. Firstly, we construct a new constraints model for the constraints relationship of simple and complex data types derived from WSDL document, in order to express the constraint relationship among data accurately. Secondly, we transform the existing test cases set built in the previous steps, and minimize the test cases as well as enhance the error detection capability of existing test cases. There are three kinds of constraints: *Cardinality* constraints, *ValueRange* constraints and Rules. The constraints models for simple data types and complex data types are defined as follows:

**Definition 5.** Data constraints model

*SimpleDataConstrain* =
 <*Cardinality*, *ValueRange*, *Rules*>

*ComplexDataConstrain* =
 <*Cardinality*, *ValueRange*, *innerRules*, *Rules*>

*Cardinality* and *ValueRange* define the constraints relationship of data attributes. *Cardinality* refers to the constraints of number, including maximum, minimum and fixed cardinality constraints, corresponding to the *minOccurs* and *maxOccurs* attributes in the WSDL file.

*ValueRange* refers to a range of constraints by object restriction, corresponding to the *Restriction* attribute in the WSDL file. *Rules* refers to the complex constraints relationship between the attribute in object restriction. According to the range, the constraints relationship can be subdivided into two types: the constraints relationship called *innerRules* for the same data between different attributes, and another one called *Rules* for the different data between attributes. The above constraints relationship can be described by the rule language Semantic Web Rule Language (SWRL) [24].

## 4. Test Data Generation Based on T Model

### 4.1. Test Process Instance

The proposed steps for initial test data generation based on the model above are as follows:

**Step 1.** At first, derive service operation and parameter information by WSDL URL, and then a formal model of type tree of Section 3.2. will be constructed by the input element of a single operation of Web Service.

**Step 2.** Generate the data constraints model of Section 3.4. according to the facet constraints relationships and user-defined rules of a single operation of Web Service.

**Step 3.** Divide the input domain by equivalence partitioning based on the formal model of type tree and data constraints model constructed by previous steps. Then select a few representative data as a test case from data subset and divide them into valid and invalid equivalence classes.

**Step 4.** Obtain built-in data types set, including *int*, *double*, *string*, *decimal* and other original input data types, and select factors boundary based on representation constraints, such as: (1) Minimum and maximum values of the data type; (2) Minimum and maximum length of String; (3) True and false of Boolean.

**Step 5.** According to the equivalence classes and boundary values for each data partition, select boundary value, abnormal value, null value, normal value and other factors, and then obtain the initial test suit completely based on integrity constraints.

## 4.2. Test Data Generation

According to the formal model of type tree, test data generation algorithm is presented as follows:

*Algorithm 2.* Initial test data generation.

```
Input: A formal model of type tree of single operation in WSDL:
        $T (N, S, B, n_r, IC, RC, EE, ED)$, data constraints set Constraints
Output: initial test data set TestData
TestData = ∅; // Initialize TestData
GetTestData(T): // Test data generation algorithm for T
        For each node 'm' in B and e (n, x, m) // Traversal each root node in the T model
                GetData (n, Constraints);
        End for;
GetData (n, Constraints): // Test data generation algorithm
        TD = ∅; // Initial test suite for sub-element in complex data
        If (Constraints = = SimpleConstraint)
        {
          While ($x \neq \emptyset$) {
                According to the domain constraints in data constraints set Constraints,
                generate test data TestData through equivalence class and boundary
                partition, and put it into factors set F;
                m = n;
                GetData (n, Constraints);}
        }
        else
        {
          For each SubElement in n
             TD.add (GetData (SubElement, Constraints))
             d = GetStructure(x) // derive data structure
             If (d = = sequence)
                {TestData = GetSequenceData (TD);
                 // Generate test data using equivalence classes and boundary partition based
                    on sequence rule and data constraint model Constraint, and put it into
                    factors set F;}
             else
                if (d = = choice)
                  {TestData = GetChoiceData (TD); // As in the above case, generate test data
                  based on Choice}
                else {TestData = GetAllData(TD) // As in the above case, generate test data
                      based on All}
        }
```

## 5. Obtain the Optimal Test Cases

According to the previous algorithm in Section 4.2., the initial test data will be generated for single operation of Web Service while the test data is redundant. Additionally, we propose a test set optimization approach based on combinatorial testing and data constraints rules to solve this problem. The workflow is shown in Figure 3.

### 5.1. Test Case Generation Based on Combinatorial Testing Through One-Test-at-a-Time

According to variable strength the combinatorial testing model proposed in Section 3.3., on the basis of the data factors F derived from initial test data from previous work, in this section test suite is obtained on the basis of combinatorial testing through One-test-at-a-time strategy. This paper defines set CombSet generated by factors set F and interaction relationship R in SUT and makes it contain the value of a collection among all the factors that need to be covered by the test suite. Through the end, the test suite is obtained by generating every test case using One-test-at-a-time strategy proposed in [21] as shown in algorithm 3.

*Algorithm 3.* One-test-at-a-time strategy.

```
Input: Set $F$ of factors and set $R$ of interaction
       relationship in a single operation in
       WSDL document
Output: Test suit based on variable strength
        combinatorial testing
Initial:
  $CombSet = \varnothing$
  Initialize T[0..0][1…$n$]; // initialize a matrix T
  as test suite;
  Generate set $CombSet$ based on $F$ and $R$;
  $Uncov CombSet = CombSet$;
While ($Uncov CombSet = \varnothing$);
  Generate a test case called test, add it into
  test suite T;
  Update $Uncov CombSet$, delete the
  combination covered by test;
End while
```

Due to the particularity of the formal model of type tree, we should generate variable strength

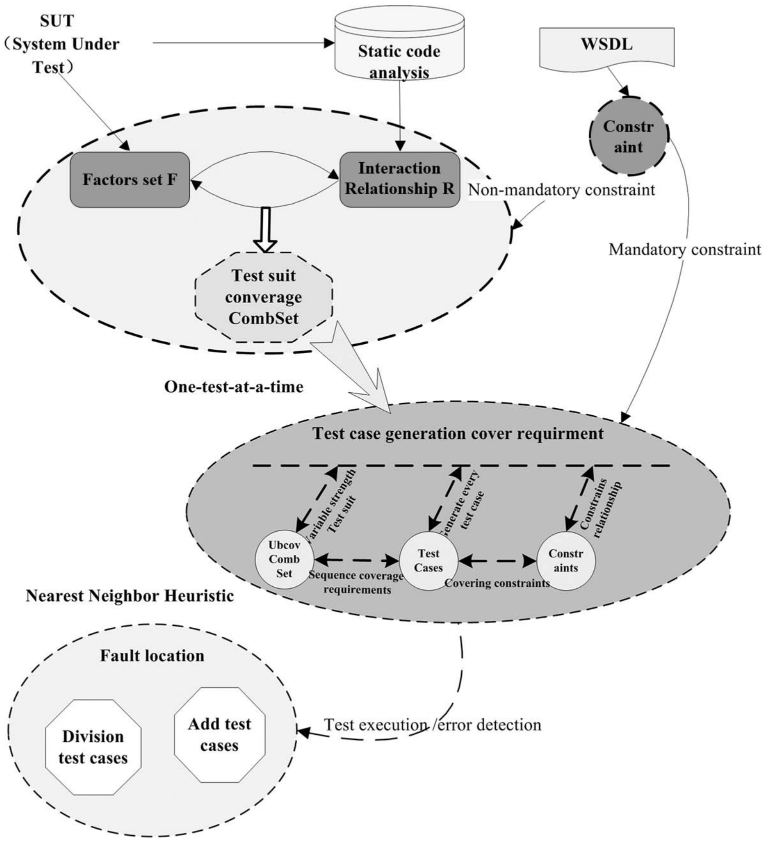

*Figure3.* The processes of test data optimization.

test suite in the complex data type firstly, afterwards regard the complex data type as a new element, and perform the test suite approach again, based on the interaction relationship. This process is shown as follows:

**Step 1.** At first, generate subfactors set and interaction relationship based on type tree formal model.

**Step 2.** Generate test suite based on combinatorial testing through One-test-at-a-time strategy for each complex data type.

**Step 3.** Regard the whole test suite generated in step 2 as a new data factor by equivalence classes and boundary values for each data partition. Then add it into set F and delete the sub -elements in complex data types in test suite $T_0$.

**Step 4.** After each set for complex data type is transferred as a new element, generates the test suite through One-test-at-a-time strategy iteratively.

### 5.2. Reform the Test Suite Based on Constraints

In this section, the test suite T created by previous section through constraints rules is reformed with the purpose of obtaining optimal test suite. Test data generation has been considered by combinatorial testing above and test cases have been dramatically minimized, but there are still some limited combinations of factors in the software which have been considered in Section 3.4. If there are strength constraints, the error detection ability of the test data would be influenced. Regarding the reduction with limited composition relationships, error detection capability of test cases will be improved. The steps are shown as follows:

**Step 1.** Firstly, obtain the limited combination of existing test suite through constraints relationship

**Step 2.** Clone the existing test cases including limited combinations

**Step 3.** Reform the positions of limited combinations for test suite from clone, and make them no longer limited

This approach will refrain from limited combinations, without affecting the coverage of valid combination by test cases.

## 6. Case Study

In this section, there is a chosen type of ship command and control system which was developed based on Service Oriented Architecture as tested service and treated as verification example of software testing. Then this section verifies the effectiveness, efficiency and practicality in data generation for single operation of Web services by our approach.

The sub-function in operational command module software is selected. After deriving the service description WSDL document, the T model and data constraints based on Section 3 are constructed at first; then the interaction between elements in T model by source code and the test cases based on combinatorial testing are generated separately; finally the final test suite based on the constraints rules between data types is achieved. Here, on the basis of comparing the proposed approach with random test case generation approach by monitoring and obtain coverage of source code, the results show that our approach gets a better code converge, specifically shown in Figure 4.

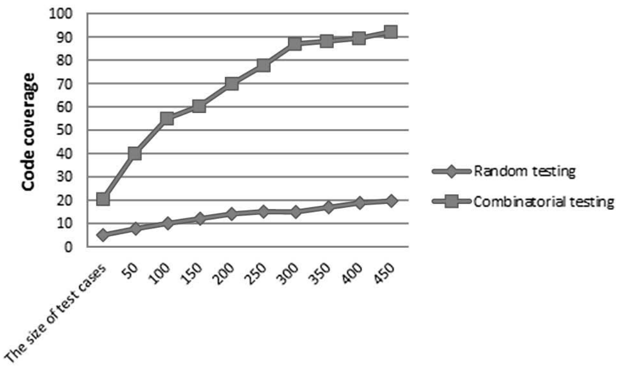

*Figure 4.* Comparison of code coverage rate between datasets.

For the validation of the advantages of test cases in size, with some existing methods, the test cases for some operations in tested service are generated; the results are shown in Figure 5. Meanwhile, Table 2 shows the detected results for multiple operations in the tested Web services with the generated test cases, including the detected faults and execution time.

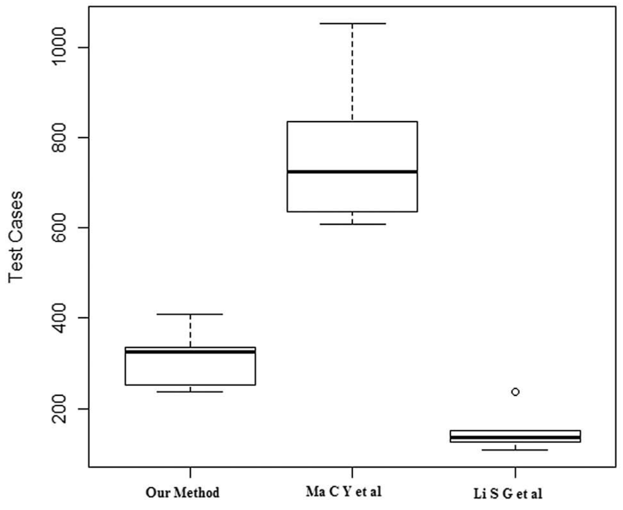

*Figure 5.* Comparison of test case size among different methods.

*Table 2.* Comparison of detected errors.

| Data Generation method | Number of errors | Error description | Execution time (s) |
|---|---|---|---|
| **Our Method** | 12 | Boundary exception, null values, combination of data processing error and so on | 230 |
| Ma *et al.* [9] | 13 | Partly Functional error | 995 |
| Li [25] | 5 | Partly boundary error | 125 |

The approach proposed by Ma *et al.* is good at error detection [9]. If the system is too complicated, it will obtain the largest amount of test cases, at the same time resulting in a combinatorial explosion which also costs the highest execution time. Although the approach proposed by Li [25] generates the least test cases, the results show that it cannot meet the test adequacy requirement with poor error detection capability. Therefore, the proposed algorithms could leverage the execution effectiveness and optimality of size of generated test suite.

## 7. Conclusion

Testing Web Services is one of the hot debates in software testing area today while test data generation for Web services has become an important part of Web services testing. In summary, this paper proposes an automatic test case generated approach, integrating combinatorial testing with data constraint rules technology. In particular, this study minimizes the test suite, ultimately solves the problem of test data redundancy and enhances the relevance of testing and improves test efficiency.

There is a workflow mentioned above for generating the test cases for Web services based on combinatorial approaches. Furthermore, testing Web services is very labor-intensive and time-consuming because SOAP message must be sent to invoke Web Service for each test case which uses a lot of resources. Accordingly, it is necessary to enhance our work by developing an automated tool to support our approach.

In the future work, more attentions should be paid to research on other levels of Web Services exhaustively, including testing the operations sequence and composite Web Services as well as further raise the automation of Web Services tests.

## References

[1] H. He (2003, Sep). What is Service-Oriented Architecture? [Online]. Available: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[2] G. Canfora and M. Di Penta, "Testing services and service-centric systems: challenges and opportunities", *IT Professional*, vol. 8, no. 2, pp. 10–17, 2006.
http://dx.doi.org/10.1109/MITP.2006.51

[3] D. Brenner *et al.*, "Strategies for the Run-Time Testing of Third Party Web Services", in *IEEE International Conference on Service-Oriented Computing an Applications (SOCA'07)*, June 2007, pp. 114–121.
http://dx.doi.org/10.1109/SOCA.2007.43

[4] A. Sinha and A. Paradkar, "Model Based Functional Conformance testing of web services operating on persistent data", in *Proceedings of 2006 workshop on Testing, analysis, and verification of Web services and application*, ACM., New York, NY, USA, 2006, pp. 17–22.
http://dx.doi.org/10.1145/1145718.1145721

[5] C. Keum *et al.*, "Generating test cases for Web services using extended finite state machine", *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, pp. 103–117, 2006. http://dx.doi.org/10.1007/11754008_7

[6] W. T. Tsai *et al.*, "Extending WSDL to Facilitate Web Services Testing," in *Proceedings of IEEE HASE*, 2002, pp. 171–172. http://dx.doi.org/10.1109/hase.2002.1173119

[7] X. Y. Bai *et al.*, "WSDL-Based automatic test case generation for Web services testing", in *Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*, Washington: IEEE Computer Society, 2005, pp. 207–212.

[8] H. Samer and M. Malcolm, "An approach for specification-based test case generation for Web Services", in *Computer Systems and Applications*, AICCSA '07, IEEE/ACS International Conference on 13–16 May 2007, pp. 16–23.

[9] C. Y. Ma *et al.*, "WSDL-Based automated test case generation for Web service", in *Proceedings of the Computer Science and Software Engineering*, Washington, 2008, pp. 731–737.

[10] J. Offutt and W. Xu, "Generating Test Cases for Web Services Using Data perturbation", *ACM SIGSOFT, Software Eng. Notes*, vol. 29, no. 5, pp. 1–10, 2004. http://dx.doi.org/10.1145/1022494.1022529

[11] W. Xu and J. Offutt, "Testing Web Services by XML perturbation", in *Software Reliability Engineering, ISSRE 2005, 16th IEEE International Symposium*, 8–11 Nov. 2005, pp.1–10.

[12] R. Siblini and N. Mansour, "Testing Web Services", in *2005 ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2005)*, Cairo, Egypt, 2005. http://dx.doi.org/10.1109/AICCSA.2005.1387124

[13] Y. Jiang, "A Method of automated test data generation for web service", *Chinese journal of computers*, vol. 28, no. 4, Apr. 2005.

[14] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing", in *Proceedings of the 27th NASA/IEEE Software Engineering Workshop*, NASA Goddard Space Flight Center, 2002, pp. 91–95.

[15] D. R. Kuhn and D. R. Wallace, "Software fault interaction and implication for software testing", in *IEEE Transactions on Software Engineering*, 2004, pp. 1–4.

[16] P. J. Schroeder *et al.*, "Comparing the fault detection effectiveness of n-way and random test suite", in *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE2004)*, Redondo Beach, Califorrnia, 2004, pp. 49–59. http://dx.doi.org/10.1109/isese.2004.1334893

[17] S. Sampath *et al.*, "Prioritizing user-session-based test cases for web application testing", in *The 1st International Conference on Software Testing, Verification, and Validation*, 2008, pp. 141–150. http://dx.doi.org/10.1109/icst.2008.42

[18] X. Yuan *et al.*, "Covering array sampling of input event sequences for automated GUI testing", in *The 22nd International Conference on Automated Software Engineering*, 2007, pp. 405–408. http://dx.doi.org/10.1145/1321631.1321695

[19] W3C. (June, 2007). Web Services Description Language (WSDL)Version 2.0 [Online]. Available: http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626

[20] Z. Y. Wang *et al.*, "Survey of combinatorial test generation", *Journal of Frontiers of Computer Science and Technology*, vol. 2, no. 6, pp. 571–588, 2008.

[21] Y. Lei, "In-parameter-order: a test generation strategy for pairwise testing", in *Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE1998)*, Washington, USA, 1998, pp. 254–261.

[22] W. Z. Yuan *et al.*, "Generating variable strength combinatorial test suite with one-test-at-a-time strategy", *Chinese Journal of computers*, vol.12, no. 35, 2012.

[23] K. J. Hou *et al.*, "Web service test Data Generation Using Interface Semantic Contract", *Journal of Software*, vol. 24, no. 9, pp. 2020–2041, 2013.

[24] I. Horrocks *et al.*, (2004). SWRL:A semantic web rule language combining OWL and RuleML [Online]. Available: http://www.w3.org/ Submission/SWRL

[25] S.-G. Li, "A method of automatic test data generation based on extended WSDL", M.S. thesis, Dept. Computer Science. Southwest University, Chongqing, China, 2010.

*Received*: June 2015
*Revised*: June 2016
*Accepted*: July 2016

*Contact addresses*:
Yin Li
Jiangsu Institute of Automation
Jiangsu, Lianyungang, Xin Pu Qu
Hailian E Rd, 42, 222006 China
e-mail: leein121999@126.com

Zhi-an Sun
Jiangsu Institute of Automation
Jiangsu, Lianyungang, Xin Pu Qu
Hailian E Rd, 42, 222006 China
e-mail: 18036671781@189.com

Jian-Yong Fang
Jiangsu Institute of Automation
Jiangsu, Lianyungang, Xin Pu Qu
Hailian E Rd, 42, 222006 China
e-mail: 18036673735@189.com

Yin Li was born in 1988. He received the MS degree from Soochow University in 2014. He is an engineer master at Jiangsu Institute of Automation. His research interests include data mining, software testing.

Zhi-an Sun was born in 1966. He received the MS degree from Beijing University of Aeronautics and Astronautics. He is a researcher at Jiangsu Institute of Automation. His research interests include software reliability, software testing.

Jian-Yong Fang was born in 1982. He received the MS degree from Beijing University of Aeronautics and Astronautics. He is a senior engineer at Jiangsu Institute of Automation. His research interests include command control, software testing.