Progressive Ranking and Composition of Web Services with Interaction Relationship

Jianjun Yuan
The Key Laboratory of Embedded System and Service Computing Ministry of Education
Tongji University, Shanghai, P.R.China
Email: 1jjyuan@tongji.edu.cn

Changjun Jiang
The Key Laboratory of Embedded System and Service Computing Ministry of Education
Tongji University, Shanghai, P.R.China
Email: cjjiang@online.sh.cn

Zuowen Jiang
The Key Laboratory of Embedded System and Service Computing Ministry of Education
Tongji University, Shanghai, P.R.China
Email: jiangzuowen@gmail.com

***Abstract*—Service-oriented computing is a new software development paradigm that allows application developers to select available services from the Internet and to form new web services. A main problem is how to efficiently determine the most appropriate web services that will be combined into an application based on their functionalities, licensing costs, and reliability. Bryce *et al.* make use of the group testing results, and then progressively select the best component WS to construct the composite service by using interaction testing. However, they consider only pair-wise (2-way) interaction among any two component WS. In really composite service, there are not always interactions among any component WS, and some component WS may need *N*-way (*N* >2) testing since there is a closer relationship among them. We extend the model of interaction testing for WS with interaction relationship and propose a greedy algorithm for constructing variable covering arrays with bias. Experimental results show that our approach can cover more important interactions among web services earlier.**

***Index Terms*—service oriented architecture, web services, covering arrays, group testing, reliability**

I. INTRODUCTION

Web Services (WS) is an emerging technology that offers a promising solution for developing distributed applications which can be accessible via the Internet. When individual web services are not able to meet complex requirements, they can be combined to create new value added composite services for those requirements [1]. The new web service can be composed dynamically at runtime, completely or partially, using existing WS available over the Internet. To assure the dependability of the new composite service, the constituent services found over the Internet must be tested in the composite service. However, the dynamic features of WS impose numerous new challenges to traditional testing techniques. For one thing the traditional 'test before delivery' pattern is no longer sufficient for the new paradigm since services are published, invoked and integrated at runtime. WS must be tested not only before, but also after, being published and composed into another web service at runtime. For another service providers may not be willing to provide the source code, and provide the user interface and functionality only. Therefore, testing can only be performed according to the specification and the interface definition of the WS in many situations [2].

A number of approaches have been proposed to ensure trustworthiness of WS and their composition. Firstly, various standards have been defined to assure interoperability, such as, Web Ontology Language for Services (OWL-S), WS Description Language (WSDL), Unified Modeling Language (UML), Business Process Execution Language for Web Services (BPEL4WS), Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI) [3, 4, 5, 6, 7, 8].

Secondly, many studies of WS testing have been made. Bloomberg divides WS testing technologies into three phases. In phase one, WS are mainly tested like the ordinary software. In phase two, the following features are involved in testing: publishing, finding, and binding capabilities of WS, the asynchronous capabilities of WS, the SOAP intermediary capability, and the quality of services. In phase three, dynamic runtime capabilities, WS orchestration testing and WS versioning will be tested [9]. Davidson also lists several testing techniques

Manuscript received
Corresponding author: Jianjun Yuan, Email: 1jjyuan@tongji.edu.cn

doi:10.4304/jsw.5.8.842-850

including: proof-of-concept testing, functional testing, regression testing, load/stress testing, and monitoring [10].

Thirdly, Tsai et al. propose a Web Services Group Testing (WSGT) technique to test a large number of WS [11]. They apply the same inputs to the WS that meet the same specification, execute the WS simultaneously, and vote the results. The WS whose outputs agree with the majority output are considered to produce a correct output. Each WS is rated according to the rate of the correct outputs. Bryce et al. make use of the group testing results, and then progressively select the best component WS to construct the composite service by using interaction testing [2].

Interaction testing is an effective method to look for defects in software systems. This method is based on the observation that a large number of faults are caused by factor (parameter) interactions in many systems. In this approach, all t-tuples of interactions in a system are incorporated into a test suite, which is called a t-way interaction test suite. The test suite's size can be reduced if we select test cases appropriately, which can also lead to reduced cost of testing.

In interaction testing, on the other hand, not all interactions are of equal importance. Some interactions may be more important since they appear more frequently. Moreover, in many applications the entire test suite is not executed because of time or budget constraints. Testers want to include certain interactions in the earliest test cases of a test suite if higher priorities are assigned to these interactions.

Bryce et al. adapt a one-test-at-a-time greedy approach to take importance of pair-wise interactions into account and use this approach to generate a set of test cases in order for WS testing [2]. However, they consider only pair-wise (2-way) interaction among any two component WS. Moreover, there are not always interactions among any component WS in really composite service. Some component WS may need N-way (N >2) testing since there is a closer relationship among them.

This paper extends the model of interaction testing for WS with interaction relationship and proposes a greedy algorithm for test suite construction that employs rankings of the constituent WS.

The remainder of this paper is organized as follows. Section II describes background and basic models. Section III develops a greedy strategy for test suite construction that employs rankings of the constituent WS, and gives a small example of building a row for a test suite. Section IV provides experimental results. Section V concludes with a discussion of the potential use of this method.

II. BACKGROUND AND DEFINITIONS

WS group testing must determine a set of composite services to construct that reveals not only errors in the individual WS, but also those that result from interactions among the WS. Therefore, once candidate WS are identified for further test, the test suite constructed provides coverage of the potential interactions.

When both the number of services and the number of candidates for each service are small, exhaustive testing is possible. But when either is large, wise selections are required to minimize the size of the test suite. Covering arrays are employed to deal with the combinatorial explosion. They have been proposed for software interaction testing [12, 13], and effective methods exist for their construction [14, 15, 16]. For WSGT, a variant of covering arrays is required to treat the different confidence levels of trust for the individual services, which means that different coverage is needed according to these confidence levels.

A. Background

Suppose that the Composite Service Under Test (CSUT) consists of n constituent WS: $S_1, S_2, \ldots, S_n$. These constituent WS are called factors of CSUT. Let $S=\{S_1, S_2, \ldots, S_n\}$ denote the set of factors, and without loss of generality. For each factor S_i, there are $|v_i|$ implementations, where $|v_i|$ is the cardinality of v_i, and v_i ($1 \le i \le n$) denotes the set of implementations of S_i. These implementations are called values of factor S_i, and may have different licensing costs and confidence levels.

Definition 1 A *covering array*, $CA(N; t, k, |v|)$, is an $N \times k$ array such that every $N \times t$ subarray contains all tuples of size t (t-tuples) at least once.

The strength of a covering array is t, which defines, for example, 2-way or 3-way interaction test suite. In our application, the k columns of this array are factors, where each factor has $|v|$ values. A t-way interaction test suite is a covering array. For instance, a CSUT has three factors (Order WS, Pay WS, and Deliver WS). For each of the factors, there are two available implementations. These implementations may have different licensing costs and confidence levels. The implementations of Order WS are denoted as values 0 and 1. Values 2 and 3 denote the implementations of Pay WS. And the implementations of Deliver WS are denoted as values 4 and 5. Four test cases (rows) in Table I cover all pair-wise interactions. Table I is an example of a pair-wise test suite.

TABLE I.
THE PAIRWISE COVERAGE IS ACHIEVED WITHIN THE FOUR TEST CASES

Test no.	Order WS	Pay WS	Deliver WS
1	0	2	4
2	0	3	5
3	1	2	5
4	1	3	4

In general, most software systems do not have the same number of values for each factor. A more general structure can be defined that allows variability of $|v|$.

Definition 2 A *mixed level covering array*, $MCA(N; t, k, (|v_1|,|v_2|,\ldots, |v_k|))$, is an $N \times k$ array on $|v|$ values, where $|v| = \sum_{i=1}^{k} |v_i|$, with the following properties: (1) Each column i ($1 \le i \le k$) contains only elements from a set C_i of size $|v_i|$. (2) The rows of each $N \times t$ subarray cover all t-tuples of values from the t columns at least once.

A shorthand notation is used to describe mixed level covering arrays by combining entries with equally sized value ranges. For instance, a CSUT has 9 factors. Four of these factors each take on 3 values, while the other 5 are binary. The pair-wise test suite of this model can be written as *MCA* (*N;* 2, $3^4 2^5$). The symbol of *k* can be dropped since it can be obtained by adding the superscripts.

B. Variable Strength Covering Array

As a factor of CSUT may interact with each other, a group of factors that have interaction with each other can be formed as a subset *u* of *S*. That means there is a |*u*|-way interaction among the factors in this subset. So for any *u*, the interaction test suite is needed to cover all valid value combinations of factors in it. If we abstract each one of all *t* interactions in CSUT as a subset of *S*, there is a collection of these subsets $U=\{u_1, u_2,\ldots, u_t\}$. A interaction test suite for this CSUT should cover all *t* interactions that associate with subsets in *U*. For instance, for a CSUT with *n* factors, a pair-wise interaction test suite should cover $|U|=n\times(n-1)/2$ different pair-wise interactions and $U=\{\{S_i, S_j\}|\ S_i, S_j\in S \text{ and } i\neq j\}$.

Definition 3 The subset $u_m \in U$ (m=1, 2,…, t) is named as an interaction coverage requirement, and the collection *U* is the interaction relationship of CSUT.

For simplicity, we assume following rules. (i) There are n_m (n_m>1) factors in each coverage requirement. (ii) For any two different coverage requirements $u_{m_1}, u_{m_2}\in U$ ($m_1\neq m_2$), u_{m_1} is not subset of u_{m_2} and vice versa. (iii) Two different factors $S_i, S_j\in S$ ($i\neq j$) interact with each other if and only if there exists a coverage requirement $u\in U$ such that $S_i, S_j\in u$.

For example, consider an CSUT with 3 factors $S=\{A, B, C\}$ and each factor has 2 values, which is shown in Fig. 1. The interaction relationship of CSUT can be described as $U=\{u_1, u_2, u_3\}$, where $u_1=\{A, B, C\}$, $u_2=\{A, C\}$ and $u_3=\{B, C\}$. That means that factors *A*, *B*, and *C* determine the Client jointly, and that factor *C* associates with factor *A* and *B* respectively.

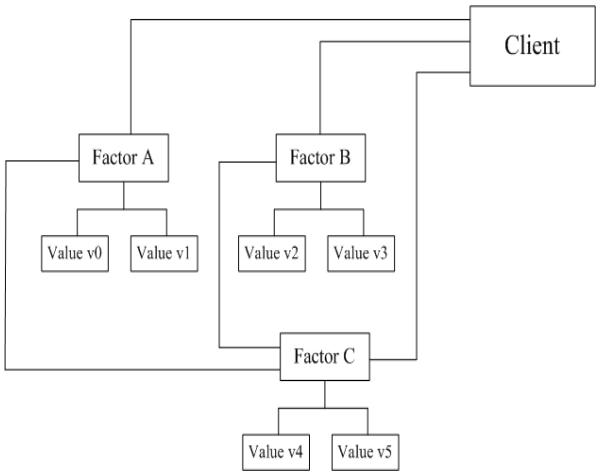

Figure 1. CUST with interaction relationship.

Definition 4 A covering array covers a coverage requirement u_m, if it contains all $|u_m|$-way interactions of factors in u_m. For an $N \times p$ array, if it covers all coverage requirements in *U*, then it is a variable strength covering array for *U* and could be denoted as *VCA*(*N*; *S*, *U*).

Therefore, the variable strength covering array for *U* should cover all combinations in set: $CombSet=\{comb|\ comb\in CombS_m\}$. And in which, the set $CombS_m$ (m=1, 2,…, t) is: $CombS_m=\{(a_{m,1}, a_{m,2},\ldots, a_{m,n_m})|a_{m,1}\in v_{m,1},\ a_{m,2}\in v_{m,2},\ldots, a_{m,n_m}\in v_{m,n_m}\}$.

C. Variable Strength Covering Arrays with Bias

In practice, some interactions are more important than others since some WS have higher confidence levels. The covering array and the variable strength covering arrays do not distinguish this, as they assume that all rows will be run as test cases and have equal priorities. When only some rows are to be used as test cases because of limited project budgets, certain test cases are more desirable than others. Both a covering array and a variable strength covering array are not sufficient models when a tester prefers to test certain interactions earlier than others.

For each value a_i in CSUT, we assume that a numerical value w_i is available as an output from the group testing described previously. $w_{i,j}$ is between 0 (the lowest ranking of trust) and 1 (the highest ranking of trust).

For a combination $comb=\{a_1, a_2,\ldots, a_p\}$ in *CombSet*, the *profit* of *comb* is:

$$w_{comb}=\sum_{1\le i<j\le p} w_i w_j+\sum_{1\le i<j<k\le p} w_i w_j w_k+\ldots+\sum_{1\le i<j<\ldots p\le p} w_i w_j \cdots w_p \quad (1)$$

Every *comb* covered by the test case contributes to the total profit, according to the reported value of trust associated with each *comb* that has not been included in a previous test case. Consider a test suite consisting of many test cases. Rather than adding the profits of each test case in the suite, we must account a profit only when a *comb* has not been covered in another test case. Each *comb* covered in a test case of the test suite may be covered for the first time by this test case, or can have been covered by an earlier test case as well. Its incremental profit is w_{comb} in the first case, and zero in the second. Then the incremental profit of the test case is the sum of the incremental profits of the *comb*s that it contains. The total profit of a test suite is the sum, over all test cases in the suite, of the incremental profit of the test cases.

Definition 5 A *l-Variable Covering Array with Bias (l-VCAB)* is a variable covering array *VCA*(*N*; *S*, *U*) in which the first *l* rows form test cases whose total profit is as large as possible.

Note that this definition should be seen as a goal rather than a requirement. Finding an *l*-VCAB is NP-hard, even if all profits for *comb*s are equal [17]. Moreover, we rarely know the value of *l* in advance. Therefore, we use the term *l*-VCAB to mean a variable covering array in which the test cases are ordered, and for every *l*, the first *l* test cases yield a large total profit.

D. Constructing Covering Arrays

The main techniques for constructing covering arrays can be classified three categories: mathematical methods, greedy algorithms, and meta-heuristic strategies. Mathematical methods for generating covering arrays usually require that each factor has the same number of values, which restrict the universality of this kind of methods. The use of orthogonal arrays belongs to this category [18, 19]. Greedy algorithms start with an empty set T and add one test at a time according to some policies like covering the most uncovered pairs [20]. Final T is a solution. The representatives of greedy algorithms include the Automatic Efficient Test Case Generator (AETG) [20], the In Parameter Order (IPO) algorithm [21, 22, 23], the Test Case Generator (TCG) [24] and the Deterministic Density Algorithm (DDA) [17]. Meta-heuristic approaches begin with individual or population covering array(s) and transform this or these covering array(s) according to various search strategies. The final answer will be found when stopping conditions are met. This kind of approaches mainly includes hill climbing [13], great deluge algorithm [13], simulated annealing [13, 25, 26], tabu search [27] and genetic algorithms [25, 28].

Of these various types of construction approaches, greedy algorithms from [17] are the most natural fit with the problem of generating VCAB since test cases are incrementally built one-row-at-a-time. Nevertheless, not all test cases in a VCAB are executed in practice as a result of time or budget constraints. Testers may only run a subset of a test suite and want to execute the test cases with the largest profit first. Greedy algorithms meet this requirement. A greedy algorithm selects values in order to provide as much coverage as possible in every iteration. Earlier rows have the largest amount of coverage. In section III, we adapt the deterministic density algorithm to generate VCAB.

III. ALGORITHMS TO CONSTRUCT VCAB

VCAB is a variation of covering arrays. In this section, we start with a review of how to generate covering arrays with a greedy approach and then explain how the greedy method is modified to generate VCAB.

A. The Framework of One-test-at-a-time Strategy

The construction of covering arrays and mixed-level covering arrays can fall into a greedy framework of one-test-at-a-time strategy and can be described at a high level. The overall goal in generating a covering array is to create a two-dimensional array in which all t-tuples associated with a specified input are covered. In such strategy, this collection is built one row at a time by fixing each factor with a value. A factor that has been assigned a value is referred to as *fixed*; one that has not, as *free*. A row may be selected from multiple candidates. When more than one candidate can be selected, several rows are constructed and one is chosen to add to the covering array. Once all t-tuples have been covered, the covering array is complete. The framework of such strategy is described as Algorithm 1.

The process of test suite (covering array) generation begins with an empty test suite. And then, test cases will be generated and added into the test suite one by one, until all combinations in set *CombSet* are covered by the test suite.

In the process of generating single test cases, it has been proven to be NP-hard in pair-wise testing that select a "best" single test case each time to cover the greatest number of uncovered combinations in *CombSet* [17]. Since pair-wise testing could be regarded as a special case of VCAB, selecting such a "best" test case in VCAB is also NP-hard. Therefore, a feasible method is to generate approximate "best" single test case by some efficient algorithm. We propose an algorithm, which is based on "density", to construct a single test case.

Algorithm 1. The framework of one-test-at-a-time strategy

```
1 Start with an empty test suite T
2 Initialize the set CombSet according to CSUT
3 While (CombSet ≠Φ)
4     Select a single test case, and add it into T
5     Modify CombSet, delete all combinations that
          covered by selected test case
6 End While
```

B. Definition of Density for VCAB

The concept of "density" is firstly defined by Colbourn et al. [17]. In that paper density is calculated in one of three ways. (i) For each new tuple that is covered in relation to fixed factors, density increases by 1.0. (ii) When no new tuples are covered, density does not change. (iii) For each new partial tuple that may be covered with free factors, a density formula calculates the likelihood of covering future tuples. For the case of pair-wise coverage, the maximum number of values for any factor is denoted as $|v_{max}|$. For factors S_i and S_j, the local density is $L_{i,j} = r_{i,j} / |v_{max}|$, where $r_{i,j}$ is the number of uncovered pairs involving a value of factor S_i and a value of factor S_j.

The definition of density in [17] is only available in pair-wise testing. Therefore we propose a new definition of "density" for generating VCAB. For each coverage requirement u_m ($1 \le m \le t$), we define the local density for VCAB as:

$$L_m = w_m / (\max_{1 \le m \le t}(w_m))^{(u_m - p_m)/u_m} \qquad (2)$$

The w_m denotes the sum of profit of available uncovered combinations, in which the values of such factors are equal to the fixed values in current test case, in set u_m. The p_m is the number of factors whose values have been fixed. When $u_m = p_m$, there will be at most one available combination. In this case, if there is an available combination, the density is the profit of this combination; else it is 0.

According to the definition of local density, the global density of CSUT could be defined as:

$$G = \sum_{m=1}^{t} L_m \quad (3)$$

When generating a single test case, we try to find a test that offers the largest incremental profit of density.

C. Algorithm to Construct Single Test Case

When constructing each single test case, an empty test case, in which values of all factors have not been fixed, will be generated firstly. At each stage, one coverage requirement, in which there is at least one factor whose value has not been fixed, will be selected for its greatest local density (assuming u_m is selected). And then, for each one of all possible uncovered combinations in u_m, if it is available in current test case, calculate the global density by assuming fixing values for factors in u_m as such available combination. The available combination that takes the greatest global density will be selected and inserted into current test case.

Iterate this process until all factors have been fixed in current test case. The pseudo-code of generating a single test case is described as Algorithm 2. We select the first one that satisfies given property to deal with tie-break. Experimental results show that there are not significant differences among various tie-break methods.

D. Algorithm Walk-Through

In this section, we illustrate the algorithm with a small example of building a row for a test suite. Consider the input in Table II. There are 5 factors in the CSUT. The values of these factors are assigned from 0 to 11. And the profit of each value is in the parentheses.

The interaction relationship of the CSUT is defined as $U = \{u_1, u_2, u_3, u_4\}$, where $u_1=\{S_1, S_2\}$, $u_2=\{S_2, S_3, S_4\}$, $u_3=\{S_4, S_5\}$, and $u_4=\{S_1, S_5\}$.

After initializing the set *CombSet* according to CSUT, we compute local density for each $u_m \in U$, where m=1, 2, 3, 4. The results are shown in Table III.

We select coverage requirement u_2 since its local density is greatest. Then for each combination $comb \in CombS_2$, global density is calculated by assuming the values of factors in u_m are fixed as *comb*. The computational results are shown in Table IV.

According to Table IV, we fix factors in u_2 as {4, 6, 8} because global density is greatest when *comb*={4, 6, 8}. Mark u_2 handled, and continue next iteration since there are also two free factors. The values of local density for u_1, u_3, u_4 are shown in Table V.

Since u_3 has the greatest local density and there is one factor, S_4, in u_3 has been fixed to value 8, for combinations {8, 9}, {8, 10}, {8, 11}, global density is calculated by assuming the values of factors in u_m are fixed as these three combinations. The computational results are shown in Table VI.

Algorithm 2. Generate a single test case

```
1  Begin with an empty test case test
2  Mark all u_m ∈ U not handled
3  While (CombSet ≠ Φ and at least one factor whose
         value has not been fixed)
4      For m=1 to t
5          If (u_m is not handled)
6              If (in u_m, at least one factor whose
                   value has not been fixed)
7                  Calculate local density for u_m ∈ U
8              End If
9          End If
10     End For
11     Select a new coverage requirement u_m with the
           greatest local density
12     Mark u_m handled
13     For each comb ∈ CombS_m
14         If (comb is available in test)
15             Calculate global density by assuming
                   the values of factors in u_m are fixed
                   as comb
16         End If
17     End For
18     Select a combination comb that takes the
           greatest global density
19     Fix factors in u_m as the selected combination
20 End While
```

TABLE II.
FACTORS, VALUES, AND PROFIT OF EACH VALUE

S_1	S_2	S_3	S_4	S_5
0(0.2)	3(0.2)	5(0.4)	7(0.1)	9(0.7)
1(0.1)	4(0.3)	6(0.5)	8(0.9)	10(0.4)
2(0.1)				11(0.8)

TABLE III.
LOCAL DENSITY FOR EACH u_m

u_1	u_2	u_3	u_4
0.0482	1	0.4578	0.1831

The results in Table VI show that global density is greatest when *comb*={8, 11}. So we fix factor S_5 to value 11. Mark u_2 and u_3 handled, and continue next loop since there is also one free factor. The values of local density for u_1, u_4 are shown in Table VII.

Select coverage requirement u_4 since its local density is greatest. Then for each available combination $comb \in CombS_4$, global density is calculated by assuming the values of factors in u_3 are fixed as *comb*. The computational results are shown in Table VIII. According to Table VIII, we arrange value 0 to factor S_1. Then the first test is generated.

TABLE IV.
GLOBAL DENSITY FOR EACH $comb \in CombS_2$

{3, 5, 7}	{3, 5, 8}	{3, 6, 7}	{3, 6, 8}
0.4637	1.7538	0.4957	1.8818
{4, 5, 7}	{4, 5, 8}	{4, 6, 7}	{4, 6, 8}
0.5373	1.9394	0.5803	2.0864

TABLE V.
LOCAL DENSITY FOR u_1, u_3, u_4

u_1	u_3	u_4
0.0589	0.8394	0.1831

TABLE VI.
GLOBAL DENSITY FOR AVAILABLE $comb \in CombS_3$

{8, 9}	{8, 10}	{8, 11}
0.8264	0.4974	0.9360

TABLE VII.
LOCAL DENSITY FOR u_1, u_4

u_1	u_4
0.0589	0.1571

TABLE VIII.
GLOBAL DENSITY FOR AVAILABLE $comb \in CombS_4$

{0, 11}	{1, 11}	{2, 11}
0.22	0.11	0.11

IV. EXPERIMENT

We have implemented both the algorithms in this paper (Profit algorithms) and the algorithms without considering profit (Non-profit algorithms) for comparison. Both algorithms are based on density. The input uses the data in Table II.

Table IX and Table X show the output of Profit algorithms and Non-profit algorithms, and the symbol '-' denotes that this place can be any value of the factor since all the combinations in *CombSet* have been covered. Both Table IX and Table X are VCA generated by using the data in Table II. We perform some experiments using other input and discover that the size of the VCA generated may vary according to the distribution of profit of values.

The amount of profit covered in each test using these two algorithms is shown in Table 11. From this table, Profit algorithms cover more profit early. Fig. 2 shows the difference in cumulative profit covered after each test.

TABLE IX.
OUTPUT OF NON-PROFIT ALGORITHMS

Test no.	S_1	S_2	S_3	S_4	S_5
1	0	3	5	7	9
2	1	3	5	8	10
3	2	4	5	7	11
4	0	4	6	7	10
5	1	4	5	8	9
6	0	3	6	8	11
7	2	3	6	7	9
8	1	4	6	8	11
9	2	-	-	-	10

TABLE X.
OUTPUT OF PROFIT ALGORITHMS

Test no.	S_1	S_2	S_3	S_4	S_5
1	0	4	6	8	11
2	0	4	5	8	9
3	0	3	6	8	10
4	1	3	5	8	11
5	1	4	6	7	9
6	2	4	5	7	11
7	2	3	6	7	10
8	2	3	5	7	9
9	1	-	-	-	10

TABLE XI.
PROFIT COVERED IN EACH TEST

Test no.	Non-profit Algorithms (%)	Profit Algorithms (%)
1	5.68	27.75
2	15.86	23.22
3	5.59	18.54
4	6.06	11.30
5	22.65	5.92
6	24.25	5.59
7	3.85	3.99
8	15.48	3.11
9	0.57	0.57

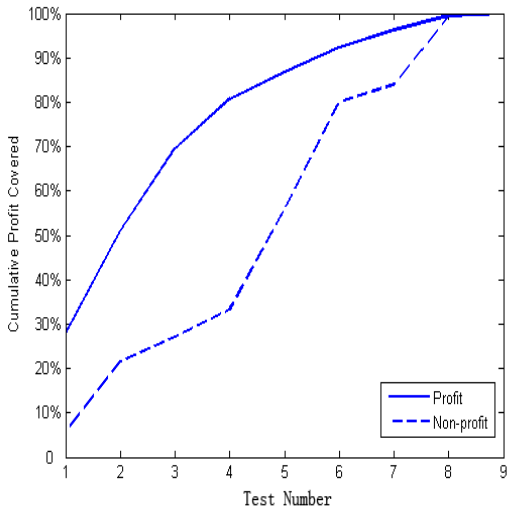

Figure 2. Cumulative profit covered using Profit algorithms and Non-profit algorithms.

The profit distribution may affect the cumulative profit

coverage. To explore this further, we perform experiments according to different profit distributions. Consider the input $16^{3}9^{4}2^{80}$ (This is a shorthand notation for a CSUT, which has 87 factors. Three of these factors each take on 16 values. Four of these factors each take on 9 values. And the other 80 are binary). The interaction relationship in this CSUT is pair-wise interaction. Three different profit distributions are as follows.

• Distribution 1 (Equal profit): All values have the same profit.

• Distribution 2 (50/50 split): Half of the profits for each factor are set to 0.8 and the other half to 0.2.

• Distribution 3 (Random): Profits are randomly distributed.

Fig. 3 shows the percentage of cumulative profit covered in the first 10 test cases for each of the three distributions. When all values have the same profit, the result is a (non-biased) variable strength covering array and the cumulative profit covered is the least in the earliest test cases. Nevertheless, when there is more different in the distribution of profits, a biased variable strength covering array can often cover more profit in the earliest test cases. For example, the distribution of 50/50 split shows the most profit coverage in the earliest test cases. This may be expected because half of the values with a profit of 0.8 comprise the majority of the profit and are rapidly covered in the early test cases. The randomly distribution is intermediate between the two extremes considered.

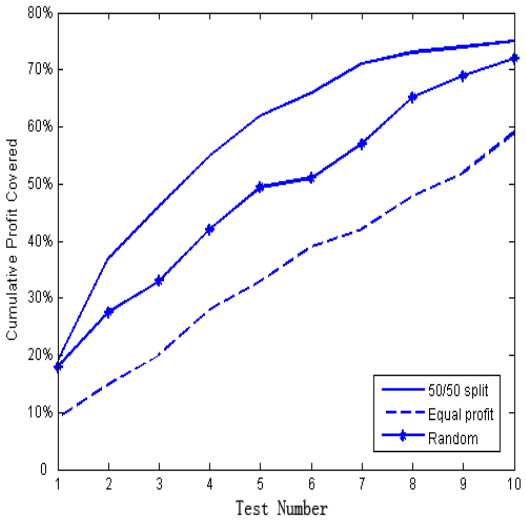

Figure 3. Cumulative profit covered under different profit distributions.

The size of the test suites produced also varies. For the three distributions, the test suites generated has 315, 327, 329 test cases, respectively. The distribution of 50/50 split generates a larger test suite than the distribution of equal profit. We perform experiments using several more examples to examine this further. Table XII shows the results in six scenarios: three have all factors with the same number of values and three are mixed level. From table XII equal profit often generate the smallest test suite.

TABLE XII.
SIZES OF TEST SUITES WITH DIFFERENT PROFIT DISTRIBUTIONS

	Equal profit	50/50 split	Random
3^{4}	9	9	13
3^{13}	19	20	20
4^{100}	46	51	53
$4^{15}3^{17}2^{29}$	38	45	48
$4^{1}3^{39}2^{35}$	27	36	39
$5^{1}3^{8}2^{2}$	20	24	27

There are mainly three kinds of threats to our findings: external, internal and construct validity.

Threats to external validity [29] are conditions that limit the ability to generalize the results of our experiments. The major external threat in this paper is our choice of input data. We cannot guarantee that these models accurately represent real CSUT.

Threats to internal validity are conditions that can affect the dependent variables of the experiment without the researcher's knowledge. There is one main threat to internal validity. Although we have verified the results of every run, we cannot be completely sure that the implementations are correct translations from pseudo-code, nor that there are not bugs in these programs.

Threats to construct validity are that we have considered both the profit covered and the size of the resulting test suite, but we have ignored other metrics that are important in some cases.

V. CONCLUSIONS

Constructing test suites for testing WS in a service-oriented architecture is a challenging task. A main challenge is how to efficiently select the most appropriate WS that will be combined into an application based on their functionalities, licensing costs, and reliability when many alternative WS for a given functional specification are provided.

Interaction testing based on density is a promising approach to test composite WS. Previous work only studies 2-way interactions between WS. Nevertheless some WS may need N-way (N>2) testing since there are closer relationship among them.

This paper extends the model of interaction testing for WS with interaction relationship and proposes a greedy algorithm for test suite construction that employs rankings of the constituent WS. The greedy method is based on density and determines a sequence of test cases to be performed. Each test case attempts to make the best incremental improvement of profit. Experimental results show that the new approach can provide the test suite in a prioritized order. In the future we will look for more methods to ensure the trustworthiness of WS.

ACKNOWLEDGMENT

This research was partially supported by Program for Changjiang Scholars and Innovative Research Team in

University, the National Natural Science Foundation of China (60534060, 90718012, and 90818023), and the National High-Tech Research and Development Plan (863) of China (2007AA01Z136, 2007AA01Z149, 2009AA01Z401, and 2009AA01Z141).

REFERENCES

[1] T. V. Xuan, H. Tsuji, and R. Masuda, "A new QoS ontology and its QoS-based ranking algorithm for web services," *Simulation Modelling Practice and Theory*, vol. 17, pp. 1378-1398, September 2009.

[2] R. Bryce, C. J. Colbourn, and Y. Chen, "Biased covering arrays for progressive ranking and composition of web services," *International Journal Simulation and Process Modeling*, vol. 3, pp. 80-87, October 2007.

[3] Ankolekar et al., "DAML-S: web service description for the semantic web," in *Proc. International Semantic Web Conference*, 2002, pp. 348–363.

[4] F. Curbera et al., "Unraveling the web services web: an introduction to SOAP, WSDL and UDDI," *IEEE Internet Computing*, vol. 6, pp. 86–93, March 2002.

[5] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The next step in web services," *Communications of the ACM*, vol. 46, pp. 29–34, October 2003.

[6] N. Milanovic, and M. Malek, "Current solutions for web service composition," *IEEE Internet Computing*, vol. 8, pp. 51–59, November 2004.

[7] W. T. Tsai et al., "Verification of web services using an enhanced UDDI server," in *Proc. IEEE WORDS*, 2003, pp. 131–138.

[8] W.T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending WSDL to facilitate web services testing," in *Proc. IEEE HASE*, 2002, pp. 171–172.

[9] J. Bloomberg, "Web services testing: beyond SOAP," Internet: http://www.zapthink.com, September 1, 2002 [March 5, 2009].

[10] N. Davidson, "Testing web services," Internet: http://www.webservices.org, October 7, 2002 [March 12, 2009].

[11] W. T. Tsai, Y. Chen, R. Paul, N. Liao, and H. Huang, "Cooperative and group testing in verification of dynamic composite web services," in *Proc. Workshop on Quality Assurance and Testing of Web-Based Applications, in Conjunction with COMPSAC*, 2004, pp. 170–173.

[12] M. B. Cohen, C. J. Colbourn, J. S. Collofello, P. B. Gibbons, and W. B. Mugridge, "Variable strength interaction testing of components," in *Proc. 27th Annual International Computer Software and Applications Conference*, 2003, pp. 413–418.

[13] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, and W. B. Mugridge, "Constructing test suites for interaction testing," in *Proc. International Conf. Software Engineering*, 2003, pp. 38–48.

[14] M. A. Chateauneuf, C. J. Colbourn, and D. L. Kreher, "Covering arrays of strength three." *Designs, Codes and Cryptography*, vol. 16, pp. 235–242, May 1999.

[15] C. Cheng, A. Dumitrescu, and P. Schroeder, "Generating small combinatorial test suites to cover input-output relationships," in *Proc. the Third International Conference on Quality Software*, 2003, pp. 76–82.

[16] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, "Augmenting simulated annealing to build interaction test suites," in *Proc. IEEE International Symposium Software Reliability Engineering*, 2003, pp. 394–405.

[17] C. J. Colbourn, M. B. Cohen, and R. C. Turban, "A deterministic density algorithm for pairwise interaction coverage," in *Proc. International Conference on Software Engineering*, 2004, pp. 245–252.

[18] R. Brownlie, J. Prowse, and M. S. Padke, "Robust testing of AT&T PMX/StarMAIL using OATS," *AT&T Technical Journal*, vol. 71, pp. 41-47, May 1992.

[19] R. Mandl, "Orthogonal latin squares: an application of experiment design to compiler testing," *Communications of the ACM*, vol. 28, pp. 1054-1058, Oct. 1985.

[20] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton. "The AETG System: An Approach to Testing Based on Combinatorial Design." *IEEE Transactions on Software Engineering*, vol. 23, pp. 437-444, July 1997.

[21] K. C. Tai and Y. Lei, "A test generation strategy for pairwise testing," *IEEE Transactions on Software Engineering*, vol. 28, pp. 109-111, January 2002.

[22] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: a general strategy for t-way software testing," in *Proc. IEEE International Conference on the Engineering of Computer-Based Systems*, 2007, pp. 549-556.

[23] L. Yu and K. C. Tai, "In-parameter-order: a test generation strategy for pairwise testing," in *Proc. 3rd IEEE International High-Assurance Systems Engineering Symposium*, 1998, pp. 254-261.

[24] T. W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Proc. IEEE Aerospace Conference*, 2000, pp. 431-437.

[25] J. Stardom, "Metaheuristics and the search for covering and packing arrays," M.A. thesis, Simon Fraser University, Canada, 2001.

[26] B. Stevens, "Transversal covers and packings," Ph.D. thesis, University of Toronto, Canada, 1998.

[27] K. Nurmela, "Upper bounds for covering arrays by tabu search," *Discrete Applied Mathematics*, vol. 138, pp. 143-152, March 2004.

[28] S. A. Ghazi and M. A. Ahmed, "Pair-wise test coverage using genetic algorithms," in *Proc. Congress on Evolutionary Computation*, 2003, pp. 1420-1424.

[29] C. Wohlin et al., *Experimentation in Software Engineering: An Introduction*. Norwell, MA: Kluwer Academic Publishers, 2000.

Jianjun Yuan received the B.S. degree in 1998 from Hubei University, China, and the M.S. degree in 2005 from Huazhong University of Science & Technology, China, both in computer science and engineering. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Technology at Tongji University, China. His research interests include web services and software testing.

Changjun Jiang received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995 and conducted post-doctoral research at the Institute of Computing Technology, Chinese Academy of Sciences, in 1997. He is a Professor with the Department of Computer Science and Technology, Tongji University, Shanghai, China. He has taken in over 20 projects supported by National Natural Science Foundation, National Key Technologies R&D Program, National Key Basic Research Developing Program, and other

key projects at provincial or ministerial levels. He has published more than 100 papers in domestic and international academic journals and conference proceedings, including *IEEE Transactions on System, Man and Cybernetics, Information Sciences* and so on. Furthermore, he has published four books (supported by Science Publishing Foundation of the Chinese Academy of Science). His current areas of research are web services, concurrent theory, Petri net, and formal verification of software.

Zuowen Jiang is currently pursuing the Ph.D. degree in the Department of Computer Science and Technology at Tongji University, China. His research interests include web services, formal methods, and model checking.