To Appear in ICWS 2010 Research Track

Taking Advantage of Service Selection: A Study on the Testing of Location-Based Web Services through Test Case Prioritization*

Ke Zhai, Bo Jiang
The University of Hong Kong
Pokfulam, Hong Kong
{kzhai, bjiang}@cs.hku.hk

W. K. Chan†
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cs.cityu.edu.hk

T. H. Tse
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

***Abstract*—Dynamic service compositions pose new verification and validation challenges such as uncertainty in service membership. Moreover, applying an entire test suite to loosely coupled services one after another in the same composition can be too rigid and restrictive. In this paper, we investigate the impact of service selection on service-centric testing techniques. Specifically, we propose to incorporate service selection in executing a test suite and develop a suite of metrics and test case prioritization techniques for the testing of location-aware services. A case study shows that a test case prioritization technique that incorporates service selection can outperform their traditional counterpart — the impact of service selection is noticeable on software engineering techniques in general and on test case prioritization techniques in particular. Furthermore, we find that points-of-interest-aware techniques can be significantly more effective than input-guided techniques in terms of the number of invocations required to expose the first failure of a service composition.**

***Keywords*—test case prioritization, location-based web service, service-centric testing, service selection**

I. Introduction

Location-based service (LBS) is indispensable in our digital life [20]. According to a keynote speech presented at ICWS 2008 [7], many interesting mobile web-based services can be developed on top of LBS [7]. In the social network application *Loopt* [8], for instance, we receive notifications whenever our friends are nearby, where each “friend” found is known as a point of interest (POI) [9][18], a specific location that users may find useful or interesting. A faulty location-based service composition may annoy us if it erroneously notifies us the presence of a stranger in another city, or fails to notify us even if a friend is just in front of us. In this paper, we use the terms “composite service” and “service composition” interchangeably.

After fixing a fault or modifying a service composition, it is necessary to conduct regression testing (i.e., retesting software following the modification) to assure that such fixing or modifications do not unintentionally change the service composition. Test case prioritization permutes a set of test cases (known as a test suite) with the aim of maximizing a testing goal [17][24]. One popular goal used by researchers is the fault detection rate, which indicates how early the execution of a permuted test suite can expose regression faults [5][6][17][23]. Prioritization does not discard any test case, and hence does not compromise the fault detection capability of the test suite as a whole.

Existing work [5][6][11][23] has studied code-coverage-based techniques for test case prioritization. Such techniques use the heuristics that faster code coverage of the application under test may lead to a faster fault detection rate. However, coverage information is not available until the corresponding test cases have been executed on the application. These techniques may approximate the required information using the coverage information achieved by the same test case on a preceding version of the application. However, such an approximation may be ineffective when the amount of code modification is large, which is the case for dynamic service composition. Furthermore, these techniques require instrumentation to collect the code coverage information. Nonetheless, such instrumentation often incurs significant execution overhead, which may prevent an LBS service from responding in a timely manner.

When performing test case prioritization on cooperative services, many researchers assume that a service composition has already been formed. In the other words, cooperative services form a service composition through static binding rather than dynamic service selection. To perform regression testing on such statically bound composite services, existing techniques [13][14] usually execute all test cases on every possible service composition, which can be costly when the pool of candidate services is large.

We observe at least two technical challenges for a technique to support the testing of dynamic service compositions: First, the exact service composition cannot be known until a dynamic service selection has been made. There is no guarantee that members in a previous service composition that executed a test case can be discovered and selected to form a service composition again for regression testing of the same test case. Second, the number of possible service compositions can be huge. Testing all combinations maximizes the fault detection capability but makes testing expensive. We argue that the testing of dynamic service compositions requires a low-cost and yet highly effective testing strategy. To address the first challenge, we bring in service selection into a test case prioritization technique. To address the second challenge, we optimize the technique by not selecting/binding a web service that has already been found to be faulty.

The main contribution of the paper is threefold: (i) We propose a novel approach to integrating service selection and

* This research is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (project no. CityU 123207 and HKU 717308) and Strategic Research Grants of City University of Hong Kong (project nos. 7008039 and 7002464).

† All correspondence should be addressed to Dr. W.K. Chan at Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong. Tel: (+852) 2788 9684. Email: wkchan@cs.cityu.edu.hk.

test case prioritization and formulate a family of black-box test case prioritization techniques. (ii) We empirically study the impact of service selection on the effectiveness of software engineering techniques in general and test case prioritization techniques in particular. (iii) We propose a suite of location-based metrics and evaluation metrics to measure the proximity/diversity between points of interests and locations in the test cases.

We organize the rest of paper as follows: We first highlight a case study in Section II that has motivated us to propose our prioritization techniques. Then, we discuss our service-selection-aware test case prioritization techniques in Section III. We present our case study on a location-based web service in Section IV. Finally, we review related work in Section V, and conclude the paper in Section VI.

II. Motivating Study

We motivate our work via an adapted location-based service composition *City Guide* as shown in Figure 1. It consists of a set of main web services (denoted by WS_i, i = 1, 2, ..., n), each binding to a map service and a Case-Based Reasoning (CBR) service, which in turns binds to a data service. The main web services are registered in a UDDI registry. Each client of *City Guide* is written in Android running on a smart phone. Our example adapts the application to let the service-selection service receive a blacklist of service URLs. The service filters out these blacklisted services and selects a remaining WS_i using its original approach. For instance, to select a web service, it applies GPS data to every discovered service and selects the one that produces the largest number of POIs.

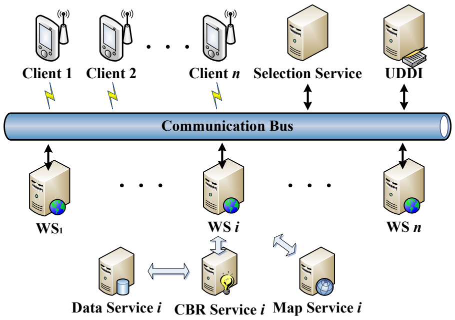


Figure 1. Architecture of *City Guide*.

On receiving GPS (i.e., location) data, a client binds itself to and invokes, say, WS_i, to obtain the best-matched POIs (e.g., hotels) based on the given locations as well as user preferences kept by the client. WS_i in turn invokes a map service using the user's location and passes the map instance and user preference data to a CBR service. Each case in the case base (stored in data service) is described by the GPS locations, user preferences, and the identity of the best-matched POI. The CBR service computes the similarity between a given query and the stored cases, and then replies to the client with a set of best-matched POIs. Service consumers may confirm the returned POIs, which will be passed back to the CBR service and retained as a new case for future reasoning.

In this scenario, services of each type may have multiple implementations. Moreover, each use of the *City Guide* application triggers a dynamic selection of concrete services of any kind followed by a dynamic formation of a service composition. However, a software service can be faulty. Our verification objective is to assure *dynamically* any service composition by using test cases that may dynamically be dispatched to verify the service composition. One may invoke all possible service compositions for all possible test cases, but such a simple approach incurs significant overheads.

A. Service Selection

Table I shows a test suite containing six ordered test cases $\langle t_1, t_2, t_3, t_4, t_5, t_6\rangle$ and their fault detection capability on four faulty web services $\{WS_1, WS_2, WS_3, WS_4\}$ in *City Guide*. A cell marked as "failed" means that the execution of the service (row) over the test case (column) produces a failure; otherwise, the test case is deemed as successful. We are going to use the example to illustrate a problem of existing techniques in testing dynamic service compositions.

Table I. Test Cases and their Results

	t_1	t_2	t_3	t_4	t_5	t_6
WS_1				failed	failed	
WS_2	failed	failed		failed		
WS_3		failed	failed			
WS_4					failed	failed

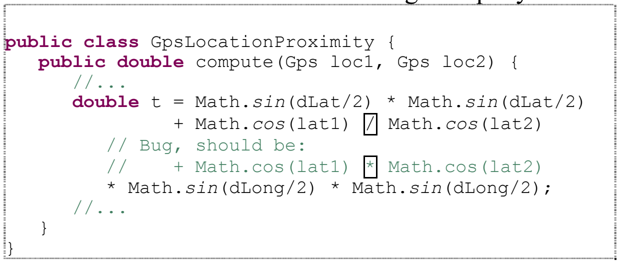
Traditional Regression Testing Techniques [17] can be ineffective in revealing faults in web services. For instance, test case prioritization techniques [24] require each test case to be executed on every web service. To do so, a technique may, in the worst case, construct 24 (= 6 × 4) service compositions. However, as we have mentioned above, there is no guarantee that a given service is re-discoverable and bound to form a required service composition so as to apply the same test case again. Indeed, in *City Guide*, service discovery and selection are performed by the application. It is hard to apply traditional testing techniques to assure *City Guide*.

Even if service compositions could be constructed, traditional techniques can still be ineffective. Let us consider a scenario: Initially, all four web services run t_1 in turn and WS_2 is detected to be faulty. Since WS_2 has been shown to be faulty, it is undesirable to select it for follow-up service compositions. Subsequently, if t_2 is run, only 3 services (WS_1, WS_3, and WS_4) need to be invoked in turn. Following the same scheme, one may easily count that the numbers of service invocations for t_3, t_4, t_5, and t_6 are 2, 2, 1, and 0. respectively, and the total number of service invocations is therefore 12. We observe that for all 12 invocations in the above scenario, only four (33%) reveal any fault. Hence, two third of the service invocations are wasted.

New Idea. We illustrate our new idea with a testing strategy that potentially reduces the number of service invocations. Let us revisit the test case execution scenario in the last paragraph. For each test case, we invoke only one service, chosen by the selection service that maintains a blacklist of services shown to be faulty. The list is initially empty. A technique passes the blacklist to the service-selection service, which discovers all four web services and picks WS_1 to be invoked because it is not blacklisted. Unfortunately, the first test case $t_1$does not reveal any fault in WS_1. The technique then applies the second test case. It passes the latest blacklist to the service-selection service, which also discovers all four web services (i.e., all services that have not been shown to be faulty) and resolves to choose WS_3 for t_2. This invocation for test case t_2 reveals that WS_3 is faulty. Hence, the technique adds WS_3 to the blacklist. The technique repeats the above procedure with WS_4 for t_3, WS_2 for t_4, WS_1 for t_1, and WS_4 for t_6 in turn. Noticeably, the total number of service invocations drops to 6. *Our insight* is that service selection has an impact on the effective application of a testing and analysis technique.

B. POI Proximity

The effectiveness of a test suite in detecting the presence of a fault in web services can be further improved by using test case prioritization [13][14]. Figure 2 shows a fault in the computation of location proximity in *City Guide*, in which a multiplication operator is mistakenly replaced by a division operator [12]. Two test cases are shown in Figure 3. Each time when the web services recommend three hotels, the client chooses the closest one as the best hotel, which, together with user's GPS location, will be added as a new case in the case base for future reasoning and query.



```
public class GpsLocationProximity {
   public double compute(Gps loc1, Gps loc2) {
      //...
      double t = Math.sin(dLat/2) * Math.sin(dLat/2)
               + Math.cos(lat1) / Math.cos(lat2)
         // Bug, should be:
         //    + Math.cos(lat1) * Math.cos(lat2)
         * Math.sin(dLong/2) * Math.sin(dLong/2);
      //...
   }
}
```

Figure 2. Faulty code with wrong operator

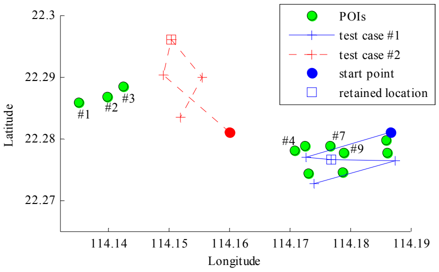


Figure 3. Test cases with different POI coverage

Consider test case #1, where the user's locations are close to the POIs. The correct distances of POIs #7 and #9 from the third GPS location `(114.1867, 22.2812)` are `0.2569 km` and `0.2545 km`, respectively, and POI #9 is the closest. However, the distances computed by the faulty code are `0.2570 km` and `0.2704 km`, respectively, and POI #7 becomes the closest POI. Although the fault only incurs a marginal error in the distance calculation, the small difference seriously affects future POI results because the user will mistakenly confirm a POI that is not optimal. As a result, test case #1 exposes a failure.

On the contrary, test case #2 does not reveal any failure. Although the distance is wrongly computed whenever the function is called, it leads only to a small error because the locations are always far away from any POIs, which does not affect the POI ranking. As a result, POIs #1, #2, and #3 are always ranked as closest.

The above example shows that the closer a GPS location sequence is in relation to the POIs, the more effective can be the sequence for detecting faults in location-based web services. To achieve better regression testing effect, we propose several POI-aware test case prioritization techniques in Section III.

III. Service-Centric Testing Technique

Based on Section II, we now present how we address the technical challenges. The first challenge is addressed by incorporating service selection into test case prioritization. For our service-centric testing technique, a test suite is executed only once. During the execution, the binding between the client and the other web services is dynamic. In particular, for each round of execution, the client program passes a blacklist to a service-selection service, which returns a selected service for the former service to construct a service composition (see Figure 4). Hence, test cases in the test suite will not necessarily be bound to the same web service, and more than one web service can be tested by each run of the same test suite.

Our service-centric testing technique is formulated as follows. Suppose that $\langle t_1, t_2, \ldots, t_j, \ldots, t_n \rangle$ is an ordered test set provided by a non-service-centric test case prioritization technique (such as in [10][14][17]), and $\Gamma(B, A)$ is a service selection function, which accepts a blacklist B of services and a set of candidate services A, and returns a service from the set $A - B$. Let B_j be the blacklist obtained before executing the test case t_j and let B_0 be an empty set. A service-centric testing technique will set $B_{j+1} = B_j$ if the test case t_j does not reveal a failure and set $B_{j+1} = B_j \cup \{\Gamma(B_j, A)\}$ if t_j reveals a failure from executing the service composition constructed from using $\Gamma(B_j, A)$ over the test case t_j.

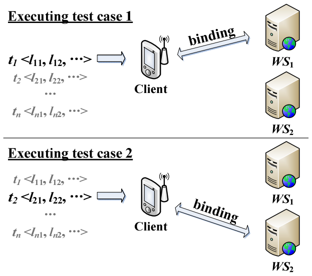

Executing test case 1
t_1 <l_{11}, l_{12}, ···>
t_2 <l_{21}, l_{22}, ···>
···
t_n <l_{n1}, l_{n2}, ···>
Client
binding
WS_1
WS_2
Executing test case 2
t_1 <l_{11}, l_{12}, ···>
t_2 <l_{21}, l_{22}, ···>
···
t_n <l_{n1}, l_{n2}, ···>
Client
binding
WS_1
WS_2

Figure 4. Our service-centric testing technique.

We next present how we address the second technical challenge for testing location-aware service compositions. In the rest of this section, we will introduce five proposed techniques using five different metrics: sequence variance

(*Var*), centroid distance (*CDist*), polyline entropy (*Entropy*), polyline distance (*PDist*), and POI coverage (*PCov*).

We categorize our proposed techniques as either input-guided or POI-aware. For the former category, we apply the concept of test case diversity as discussed in [2]. Indeed, our previous work [22] has demonstrated that the more diverse a context sequence is, the more effective they will be in fault detection. For location-based services, the input-guided techniques prioritize a test suite in descending order of the diversity of locations in test cases. For the latter category, following our observation in Section II, POI-aware techniques prioritize test cases that are closer to POIs or cover more POIs.

We first formulate some concepts for ease of discussion. Let $T = \{t_1, t_2, \ldots, t_n\}$ be a test suite with n test cases. Each test case $t = \langle l_1, l_2, \ldots, l_{|t|} \rangle$ is a sequence of GPS locations. Every GPS location $l_{i,j} = (long_i, lat_i)$ is an ordered couple of real numbers representing the longitude and latitude of a location. POIs are a set $P = \{p_1, p_2, \ldots, p_m\}$ of m GPS locations. Each POI is also denoted by an ordered couple $p_k = (long_k, lat_k)$. The objective is to prioritize T into an ordered test sequence $S = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_n} \rangle$, where $\langle i_1, i_2, \ldots, i_n \rangle$ is a permutation of $\langle 1, 2, \ldots, n \rangle$.

In our proposed techniques, the test sequence S is determined by sorting the test cases t_i in T according to the value of the quantitative metric M over t_i. This can be described as a sorting function $S = sort(T, M)$. Typically, a sorting function is defined either in ascending order ($sort_{asc}$) or descending order ($sort_{desc}$). Moreover, our proposed techniques use different quantitative metrics to guide the sorting progress to obtain a desirable value of a goal function $g(G, sort(T, M))$, which indicates how well S scores with respect to G. Without loss of generality, let us assume that a larger value of g indicates a better satisfaction of G by S. M is either a POI-aware metric $M_p(t, P)$ or an input-guided metric $M_s(t)$. The following subsections describe our proposed techniques in more detail.

A. Sequence Variance (*Var*)

Sequence variance is an input-guided metric to measure the variations in a sequence. It is defined as the second-order central moment of the sequence:

$$Var(t) = \frac{1}{|t|} \sum_{i=1,2,\ldots,|t|} \left\| l_i - \bar{l} \right\|^2$$

where $t = \langle l_1, l_2, \ldots, l_{|t|} \rangle$ denotes a test case and $\bar{l} = (\sum l_i)/|t|$ is the centroid of all GPS locations in the sequence. Intuitively, a larger variance indicates more diversity, and hence the sorting function $sort_{desc}(T, Var(t))$ is used for this technique.

B. Polyline Entropy (*Entropy*)

A test case $t = \langle l_1, l_2, \ldots, l_{|t|} \rangle$ consists of a sequence of GPS locations. If we plot these locations in a two-dimensional coordinate system and connect every two consecutive locations with a line segment, we obtain a polyline with $|t|$ vertices and $|t| - 1$ segments. *Polyline entropy* is a metric gauging the complexity of such a polyline. We adapt this metric from the concept of entropy of a curve.

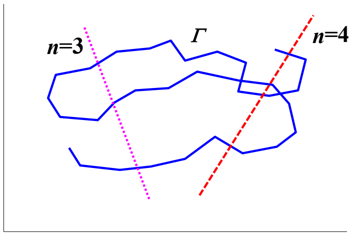


Figure 5. Illustration of the number of intersect points.

The entropy of a curve comes from the thermodynamics of curves developed by Mendès France [15] and Dupain [4]. Consider a finite planar curve Γ of length L_Γ. Let Cr be the convex hull of Γ, and C_Γ be the length of Cr's boundary. Let D be a random line, and P_n be the probability that D intersects with Γ at n points, as illustrated in Figure 5. According to [15], the entropy of the curve is given by

$$H[\Gamma] = -\sum_{n=1}^{\infty} P_n \log P_n \qquad (1)$$

By the classical computation in [15], one can easily obtain (see refs. [4][15]) the function $H(\Gamma)$ that computes the entropy of a planar curve Γ as

$$H(\Gamma) = \log\left(\frac{2L_\Gamma}{C_\Gamma}\right)$$

We follow the concept of entropy of a curve in [4][15] and compute the entropy of a test case using the function

$$Entropy(t) = \log\left(\frac{2L_t}{C_t}\right)$$

where L_t is the length of the polyline represented by t, and C_t is the boundary length of the convex hull of the polyline. A test case with higher entropy contains a more complex polyline. We sort the test cases in descending order of their entropies, that is, we use $sort_{desc}(T, Entropy(t))$.

C. Centroid Distance (*CDist*)

Centroid distance represents the distance from the centroid of a GPS location sequence to the centroid of the POIs. Because POI information is used in the computation, centroid distance is a POI-aware metric. It directly measures how far a test case is from the centroid of the POIs. We formulate this metric as

$$CDist(t, P) = \left\| \bar{l} - \bar{p} \right\|$$

where $\bar{p} = (\sum p_i)/m$ is the centroid of all POIs. The sorting function used with *CDist* is $sort_{asc}(T, CDist(t, P))$.

D. Polyline Distance (*PDist*)

Similar to *Entropy*, we may regard each test case as a polyline whose vertices are GPS locations. The polyline distance measures the mean distance from all POIs to this polyline. Let $dist(p, t)$ denote the distance from a POI $p = (long, lat)$ to a polyline $t = \langle l_1, l_2, \ldots, l_{|t|} \rangle$. The polyline distance *PDist*(t, P) of a test case t is give by

$$PDist(t, P) = \frac{\sum_{i=1,2,\ldots,|P|} dist(p_i, t)}{|P|}$$

Similar to *CDist*, we use $sort_{asc}(T, PDist(t, P))$ as the sorting function for *PDist*.

E. *POI Coverage* (*PCov*)

POI coverage evaluates the impact of POIs on each test case. To compute the *PCov* value of a test case t, we first compute the distance $dist(p_i, t)$ from each POI p_i to the polyline represented by t. Then, we use a *threshold* value α to classify whether a POI is *covered* by the polyline, by checking whether the distance $dist(p_i, t)$ is no greater than α. Hence, the *PCov* metric is given by

$$PCov(t, P) = \sum_{i=1,2,\dots,|P|} f\big(dist(p_i, t)\big)$$

where

$$f(dist) = \begin{cases} 1 & if\ dist \le \alpha \\ 0 & otherwise \end{cases}$$

Here, we use the sorting function $sort_{desc}\big(T, PCov(t, P)\big)$.

We summarize all the proposed techniques in Table II.

TABLE II. SUMMARY OF PROPOSED TECHNIQUES

Acronym	Type	Description
Var	Input-guided	Sort in descending order of the variance of the GPS location sequence
Entropy		Sort in descending order of the entropy of the polyline represented by each test case
CDist	POI-aware	Sort in ascending order of the distance between the centroid of the GPS locations and the centroid of the POIs
PDist		Sort in ascending order of the mean distance from the POIs to the polyline
PCov		Sort in descending order of the number of POIs covered by each test case

IV. CASE STUDY

In this section, we evaluate the effectiveness of our black-box testing prioritization techniques for location-based web services through a case study.

A. *Research Questions*

In this section, we present our research questions.

***RQ*1:** *Is the proposed service-centric testing technique significantly more cost-effective than traditional non-service-centric techniques?* The answer to this question will tell us whether incorporating services selection will have an impact on the effectiveness of software engineering techniques in general and test case prioritization techniques in particular.

***RQ*2:** *Is the diversity of locations in test cases a good indicator for early detection of failures?* The answer to this question will tell us whether prioritization techniques based on the diversity of locations in test cases can be promising.

***RQ*3:** *Is the proximity of locations in test cases in relation to POIs a good indicator for early detection of failures?* The answer to this question will tell us whether test case prioritization techniques based on such proximity is heading towards a right direction.

B. *Subject Pool of Web Services*

In the experiment, we used a realistic location-based service composition *City Guide*, which contains 3289 lines of code. (This will be the only subject used in our empirical study, as the implemented codes of other backend services are not available to us.)

We treat the given *City Guid*e as a “golden version”. We used MuClipse [12] to generate a pool of faulty web services and followed the procedure in [12] to eliminate mutants that are unsuitable for testing experiments. All the remaining 35 faulty web services constituted our subject pool. We applied our test pool (see Section IV.E) to these faulty web services. Their average failure rate is 0.0625.

C. *Experimental Environment*

We conduct the experiment on a Dell PowerEdge 1950 server running Solaris UNIX and equipped with 2 Xeon X5355 (2.66Hz, 4 core) processors and 8GB memory.

D. *Effectiveness Metrics*

Some previous work uses the average percentage of fault detection (APFD) [6] as the metric to evaluate the effectiveness of a test case prioritization technique. However, the APFD value depends on test suite size. For example, appending more test cases to an ordered test suite will jack up the probability to a value quite close to 1. Hence, it is undesirable to use APFD when test suites are large, but regression test suites in many industrial settings are usually large in size.

We propose to use another metric, the Harmonic Mean (HM), which is independent of the test suite size. HM is a standard mathematical average that combines different rates (which, in our case, is the rate of detecting individual faults) into one value. Let T be a test suite consisting of n test cases and F be a set of m faults revealed by T. Let TF_i be the first test case in the reordered test suite S of T that reveals fault i. Then, the harmonic mean of TF is given by

$$HM_{TF} = \frac{m}{\frac{1}{TF_1} + \frac{1}{TF_2} + \dots + \frac{1}{TF_m}}$$

We also propose to use another measure to evaluate the cost of detecting each fault. As service invocations can be expensive, a technique should aim at lowering the number of service invocations for failure detection. Suppose the number of service invocations for detecting fault i is SI_i. We propose the use the harmonic mean of SI, given by

$$HM_{SI} = \frac{m}{\frac{1}{SI_1} + \frac{1}{SI_2} + \dots + \frac{1}{SI_m}}$$

Although both HM_{TF} and HM_{SI} measure the fault detection rate, HM_{SI} is arguably more accurate than HM_{TF} because HM_{SI} reflects the actual number of service execution needed to reveal an average fault, whereas HM_{TF} reflects the number of test cases executed to reveal a fault. The latter ignores the possible dynamic binding of services and is an indirect measure of the testing effort.

E. *Experiment and Discussions*

As introduced in Section II, our subject program *City Guide* is a location-based service composition. We used a regression test pool containing 2000 test cases, each of which was a GPS location sequence. For each test case, we used the POIs returned by the golden version over the last location in a test case as the test oracle. We used the POIs extracted from the test oracles to populate the case base.

We proposed five prioritization metrics: *Var*, *Entropy*, *CDist*, *PDist*, and *PCov*. Together with random ordering, therefore, there are six techniques, each of which can be combined with service selection (denoted by service-centric) or be used alone (denoted by non-service-centric). Thus, there are twelve techniques in total. To show the average performance of our techniques on different test suites, we randomly constructed 50 test suites from the test pool. Each test suite contained 1024 test cases. Each technique was evaluated on all test suites to obtain an average. In the experiment, each service-centric technique used the adapted service-selection service of *City Guide* to implement Γ (see Section III for details).

We compute the HM_{TF} distribution for all techniques, and group the results into two box-and-whisker plots (Figures 6 (a) and (c)) depending on whether a technique is service-centric. For each plot, the *x*-axis represents the prioritization metric used (or random ordering) and the *y*-axis represents the HM_{TF} distributions for all test suites. The horizontal lines in the boxes indicate the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, then the median values of the two groups differ at a significance level of 5%. Similarly, we calculate the HM_{SI} distribution and draw two plots as shown in Figures 6 (b) and (d), where the *y*-axis represents the HM_{SI} distribution for all test suites.

For service-centric techniques, we also conduct multiple comparison analysis [11] to find those techniques whose means differ significantly from others. The distributions of HM_{TF} and HM_{SI} values for each technique are shown in Figures (e) and (f), respectively, as a horizontal line with a dot in the center, which denotes the mean value. If the lines corresponding to two techniques do not overlap, then their mean values are different at a significance level of 5%.

*1) Answering RQ*1. By comparing Figures 6 (b) and (d), we observe that service selection remarkably reduces the number of service invocations. Take the metric *Var* as an example. The service-centric technique that incorporates service selection achieves 126.88 in terms of the median HM_{SI}. However, the median HM_{SI} for the non-service-centric counterpart is 270.97. Thus, service selection leads to a 53.18% reduction of service invocations, which is an encouraging improvement. Similarly, the reductions for random ordering, *Entropy*, *CDist*, *PDist*, and *PCov* are 24.24%, 0.51%, 29.69%, 40.61%, and 23.88%, respectively. The average improvement is 28.69%, which is significant. On the other hand, the variance is large.

Comparing Figures 6 (a) and (c), we observe that the number of test cases that exposes a fault increases from using a non-service-centric version to a service-centric version of the same technique. However, as discussed in the last subsection, HM_{SI} is more accurate and the improvement of HM_{SI} through service selection is large. It further indicates that the use of traditional idea to count test cases as an effectiveness metric for service-oriented testing may not be helpful.

Hence, we can answer *RQ*1 that the proposed service-centric techniques are, on average, significantly more cost-effective in detecting faults than the non-service-centric counterparts.

*2) Answering RQ*2. We observe from Figures 6 (a), (b), (e), and (f) that, in general, the two input-guided techniques are significantly more effective than random ordering. In particular, we see from Figure 6 (b) that the median HM_{SI} values of *Var* and *Entropy* are 126.88 and 149.58, respectively, and the median HM_{SI} value of random ordering is 176.83. From Figure 6 (f), the mean HM_{SI} values of random ordering, *Var*, and *Entropy* are 182.24, 141.79, and 144.54, respectively. These figures indicate that, compared with random ordering, *Var* or *Entropy* has the potential to reduce the average number of service invocations. Moreover, *Var* outperforms *Entropy* in that *Var* leads to fewer service invocations than *Entropy*. Hence, we can answer *RQ*2 that the diversity of locations in test cases can be a good indicator to guide test case prioritization to detect failures earlier than random ordering.

*3) Answering RQ*3. We observe from the box-and-whisker plots in Figures 6 (a) and (b) that the three POI-aware techniques (*CDist*, *PDist*, and *PCov*) are significantly better than random ordering and the two input-guided techniques (*Var* and *Entropy*) at a significance level of 5%.

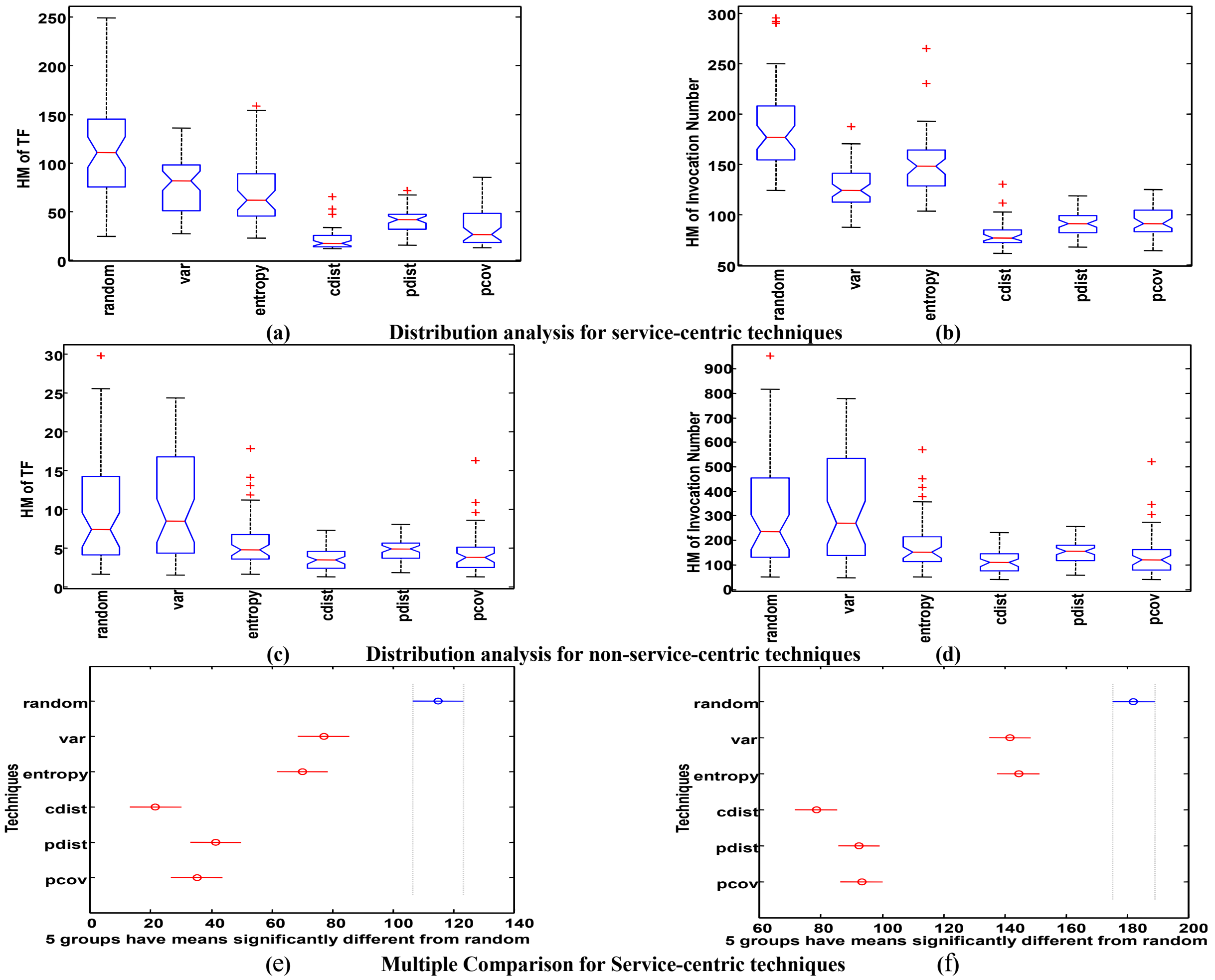
If we further examine the results of multiple comparisons [11] in Figures 6 (e) and (f), there is no overlap of POI-aware techniques with *Var*, *Entropy*, or random ordering in the respective notches. It indicates that the POI-aware techniques can be more effective than *Var, Entropy*, and random ordering in terms of mean values at a significance level of 5%. Among all techniques, *CDist* is the most effective metric to guide test case prioritization and the difference between *CDist* and each of other techniques is statistically significant. Moreover, based on Figure 6 (d), *CDist* achieves 78.4 in terms of the mean HM_{SI}, which is substantially smaller than that of random ordering (176.83). Based on the analysis, the proximity of locations in test cases in relation to POIs can be promising in guiding the detection of failures in location-based web services.

F. Summary

Our empirical results provide a piece of evidence that service selection does carry impacts on the effectiveness of software engineering techniques. According to the case study, on average, it helps improve the effectiveness of test case prioritization to assure web services remarkably.

We also observe that use of the proximity/diversity of locations is promising in guiding a testing technique to detect failures in location-based web services notably and is significantly more cost-effective than random ordering. Furthermore, the use of the locations of POIs captured by test cases can be more effective than using test inputs only. On average, POI-aware techniques detects the first failure of each fault in a location-based web service by invoking web services much fewer number of times than random ordering of test cases.

Owing to the probabilistic nature of service selection in *City Guide*, some faults may fail to be exposed. Encouragingly, we observe empirically that a fault is missed by



(a) Distribution analysis for service-centric techniques (b)

(c) Distribution analysis for non-service-centric techniques (d)

(e) Multiple Comparison for Service-centric techniques (f)

Figure 6. Experimental results.

random ordering in just one test suite out of 50, and none of the other techniques fail to expose any fault using any test suite.

We have repeated our experiment on smaller test suites and found that, although the use of a smaller test suite tends to miss more faults, the total number of missed faults is still small. For example, for test suites of size 256, on average, our proposed techniques detect at least 80% of all faults. Owing to the page limit, we do not include the detailed results on smaller test suites in this paper.

V. Related Work

Many existing test case prioritization techniques are coverage-based. For instance, Wong et al. [23] propose to combine test suite minimization and test case prioritization to select test cases based on the additional cost per additional coverage requirement. Srivastava et al. [19] propose to compare different program versions at machine code level and then prioritize test cases to cover the modified parts of the program maximally. Walcott et al. [21] propose a time-aware prioritization technique based on a generic technique to permute test cases under given time constraints. Li et al. [11] propose to apply evolutionary algorithms for test case prioritization with the goal of increasing the coverage rate.

Researchers have also investigated the challenges in regression testing of service-oriented applications. Mei et al. [14] propose a hierarchy of test case prioritization techniques for service-oriented applications by considering different levels of services including business process, XPath, and WSDL specifications. In [13][14], they also study the problem of black-box test case prioritization of service-oriented applications based on the coverage information of WSDL tags. Different from their work that explore XML message structure exchanged between services to guide prioritization, we utilize the distributions of location and POI information to guide prioritization and do not need to analyze communication messages, which are linked to location-based software cohesively.

Adaptive random testing [2][3] improves the performance of random test case generation by evenly spreading test cases across the input domain. Jiang et al. [10] proposed a family of adaptive random test case prioritization techniques that spread the coverage achieved by any prefix of a prioritized sequence of test cases as evenly as possible to increase the fault detection rate.

Locations-based web service is a popular application that can benefit both mobile network operators and end users [16]. There are many standards [1] and techniques for location-based web services. In future work, one may generalize our techniques so that they will be applicable to a broader range of location-based web services.

VI. Conclusion

The testing of dynamic service compositions must solve the problem that a dynamically selected service may not be selected and bound in another execution of the same test case. Moreover, the number of possible service compositions can be huge. Both problems need to be addressed by non-traditional testing ideas and the understanding of the impact of service selection on software engineering techniques is vital.

To tackle these two issues, we have proposed to integrate service selection in test case prioritization to support regression testing. Furthermore, we have also proposed a family of black-box service-centric test case prioritization techniques that guide the prioritization based on Point of Interest (POI) information. Our case study on a medium-sized location-based web service *City Guide* has shown that service selection significantly improves the effectiveness of regression testing. The result demonstrates that service selection has a large impact on the effectiveness of software engineering techniques in general and test case prioritization techniques in particular. Moreover, POI-aware prioritization techniques are much more effective than random ordering. The experiment has also shown that the use of proximity or diversity of locations, particularly the POI-aware properties, can be promising in cost-effectively detecting failures in location-based web services. Our future work includes incorporating advanced service selection strategies to a wider class of software engineering techniques.

References

[1] P. M. Adams, G. W. B. Ashwell, and R. Baxter. Location-based services: an overview of the standards. *BT Technology Journal*, 21 (1): 34–43, 2003.

[2] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse. Adaptive random testing: the ART of test case diversity. *Journal of Systems and Software*, 83 (1): 60–66, 2010.

[3] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Advances in Computer Science: Proceedings of the 9th Asian Computing Science Conference* (*ASIAN* 2004), volume 3321 of Lecture Notes in Computer Science, pages 320–329. Springer, Berlin, Germany, 2004.

[4] Y. Dupain, T. Kamae, and M. Mendès France. Can one measure the temperature of a curve? *Archive for Rational Mechanics and Analysis*, 94 (2): 155–163, 1986.

[5] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering* (*TSE*), 28 (2): 159–182, 2002.

[6] S. G. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Software Quality Control*, 12 (3): 185–210, 2004.

[7] C. Gonzales. Keynote 3: mobile services business and technology trends. 6th International Conference on Web Services (ICWS 2008). Beijing, 2008. Abstract available at http://conferences.computer.org /icws/2008/ Keynotes.html #Keynote3.

[8] http://www.loopt.com/. Last accessed February 2010.

[9] C.-W. Jeong, Y.-J. Chung, S.-C. Joo, and J.-W. Lee. Tourism guided information system for location-based services. In *Advanced Web and Network Technologies, and Applications*, volume 3842 of Lecture Notes in Computer Science, pages 749–755. Springer, Berlin, Germany, 2006.

[10] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse. Adaptive random test case prioritization. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2009*), pages 233–244. IEEE Computer Society Press, Los Alamitos, CA, 2009.

[11] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *IEEE TSE*, 33 (4): 225–237, 2007.

[12] Y.-S. Ma, J. Offutt, and Y.-R. Kwon. MuJava: a mutation system for Java. In *Proceedings of the 28th International Conference on Software Engineering* (*ICSE 2006*), pages 827–830. ACM Press, New York, NY, 2006.

[13] L. Mei, W. K. Chan, T. H. Tse, and R. G. Merkel. Tag-based techniques for black-box test case prioritization for service testing. In *Proceedings of the 9th International Conference on Quality Software* (*QSIC 2009*), *pages* 21–30. IEEE Computer Society Press, Los Alamitos, CA, 2009.

[14] L. Mei, Z. Zhang, W. K. Chan, and T. H. Tse. Test case prioritization for regression testing of service-oriented business applications. In *Proceedings of the 18th International Conference on World Wide Web* (*WWW 2009*), pages 901–910. ACM Press, New York, NY, 2009.

[15] M. Mendès France. Les courbes chaotiques. *Le Courrier du Centre National de la Recherche Scientifique*, 51: 5–9, 1983.

[16] B. Rao and L. Minakakis. Evolution of mobile location-based services. *Communications of the ACM*, 46 (12): 61–65, 2003.

[17] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE TSE*, 27 (10): 929–948, 2001.

[18] W. Schwinger, Ch. Grün, B. Pröll, and W. Retschitzegger. A light-weight framework for location-based services. In *On the Move to Meaningful Internet Systems 2005: OTM Workshops*, volume 3762 of Lecture Notes in Computer Science, pages 206–210. Springer, Berlin, Germany, 2005.

[19] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA 2002*), pages 97–106. ACM Press, New York, NY, 2002.

[20] S. Steiniger, M. Neun, and A. Edwardes. Foundations of location based services. *CartouCHe Lecture Notes on LBS, version 1.0.* Department of Geography, University of Zurich, Switzerland, 2006.

[21] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. TimeAware test suite prioritization. In *Proceedings of the 2006 ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA 2006*), pages 1–12. ACM Press, New York, NY, 2006.

[22] H. Wang and W. K. Chan. Weaving context sensitivity into test suite construction. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2009*). Pages 610−614, IEEE Computer Society Press, Los Alamitos, CA, 2009.

[23] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proceedings of the 8th International Symposium on Software Reliability Engineering* (*ISSRE 1997*), pages 264–274. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[24] L. Zhang, S.-S. Hou, C. Guo, T. Xie, and H. Mei. Time-aware test-case prioritization using integer linear programming. In *Proceedings of the 2009 ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA 2009*), pages 213–224. ACM Press, New York, NY, 2009.