Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study

Ke Zhai, Student Member, IEEE, Bo Jiang, Member, IEEE, and W.K. Chan, Member, IEEE

Abstract—Location-based services (LBS) are widely deployed. When the implementation of an LBS-enabled service has evolved, regression testing can be employed to assure the previously established behaviors not having been adversely affected. Proper test case prioritization helps reveal service anomalies efficiently so that fixes can be scheduled earlier to minimize the nuisance to service consumers. A key observation is that locations captured in the inputs and the expected outputs of test cases are physically correlated by the LBS-enabled service, and these services heuristically use estimated and imprecise locations for their computations, making these services tend to treat locations in close proximity homogenously. This paper exploits this observation. It proposes a suite of metrics and initializes them to demonstrate input-guided techniques and point-of-interest (POI) aware test case prioritization techniques, differing by whether the location information in the expected outputs of test cases is used. It reports a case study on a stateful LBS-enabled service. The case study shows that the POI-aware techniques can be more effective and more stable than the baseline, which reorders test cases randomly, and the input-guided techniques. We also find that one of the POI-aware techniques, \( cdist \), is either the most effective or the second most effective technique among all the studied techniques in our evaluated aspects, although no technique excels in all studied SOA fault classes.

Index Terms—Regression testing, location-based services, black-box metrics, test case prioritization, point-of-interest

1 INTRODUCTION

Location-based services (LBS) [27], [32] are services enhanced with positional data [6]. Applications enabled with LBS can both augment their outputs with highly relevant location-related information and avoid overloading users with an excessive amount of less relevant information. As such, LBS is frequently named as a key enabler across different application domains [6], [27]. Point of interest (POI) [23], [29], [36] is an important class of location-augmented output. Many POI repositories [26] have been published on the web.

For instances, advanced GPS navigators such as Garmin’s products [24] analyze the GPS sequences of vehicles and POIs nearby to predict the possible ways and relevant traffic information to reach the destinations. Some vehicle tracking systems [34] monitor GPS data sequences and analyze them with respect to some POIs (e.g., landmarks) to estimate, for example, whether a bus adheres to a target bus schedule. Other popular examples include recommendation services (e.g., Google Places (Mobile) [11]) that allow users to indicate their preferences on the provided POIs to restrict the types of information to be displayed to the users.

Like other services, a POI-enabled application may evolve dynamically. For instance, when one of its binding subservices is temporarily inaccessible or has been obsolete, the service may require discovering services for replacement, selecting the suitable substitutes, and binding to a final candidate. However, assuming that the service is compatible with the final candidate is a huge threat to the use of the service. For instance, the service may contain service discovery faults that cause it to miss locating good candidates for replacement, or may contain execution faults that cause the application outputs failed to augment with a correct set of POIs.

Testing is a popular approach to detecting failures. Broadly speaking, there are field-based testing and laboratory-based testing [17], differentiating by whether the computing environment of the service under test is the same as the operating environment.

On the one hand, the computing environment of a service at a service deployment site may not be the same as that at the service development site. On the other hand, a complete simulation of the former environment at the development site is impractical. To safeguard a (deployed) service to work properly with its binding subservices (e.g., right after each dynamic binding), the use of field-based testing at a given site is desirable.

To check whether a service still adheres to its previous established behavior after evolution, one may execute the service over a set of regression test cases (a.k.a. a regression test suite) to determine whether this service correctly handles the scenarios specified by the test cases. If there is
any unexpected change in the service behavior, a follow-up maintenance of the service implementation or further service discovery and binding can be activated. It may trigger a new round of regression testing.

A real-world regression test suite can be large, and executing it may take many hours [14]. To reduce the time cost, a subset of a given regression test suite may be selected with the aim of minimally fulfilling an objective adequacy criterion [19], [42]. However, not all industrial settings apply such strategies. For instance, in the open-source community, developers often add new test cases to the regression test suite of an application to safeguard the application from the fixed faults, and some test cases may execute along the same program path. Nonetheless, such state-of-the-practices further prolong the execution of the whole regression test suite.

Test case prioritization permutes a given test suite with the view of maximizing certain goals [28], [41]. (It does not generate any new test suite.) Such a goal can be the rate of fault detection (e.g., APFD [10]), which measures how fast a permuted test suite exposes all the anomalies detectable by the same test suite [9], [28], [41]. Test case prioritization discards no test case, and hence does not compromise the ability of the test suite to expose all detectable anomalies.

Many test case prioritization techniques are coverage-based [9], [15]. They require runtime monitoring, such as profiling the coverage of an execution trace for a test case from the service runtime for further test analysis. Such techniques heuristically assume that a test suite achieving a faster rate of code coverage on an application before evolution can also achieve a faster rate of fault detection in the same application after evolution. Although the relationship on the execution trace sets between the two versions of the same application can be arbitrary in theory, yet such a heuristics was shown to be cost effective in previous empirical studies on various benchmarks [9], [15]. Nonetheless, they require code instrumentation or dynamic collection of the execution statistics, which incurs additional runtime overheads. They could be unattractive if an application is subject to certain service-level agreements when binding to its subservices. For instance, such an overhead may violate the quality of service (QoS) property of the application that satisfies the required service-level agreements. Even though an execution trace can be profiled, yet the application may use a different set of QoS values for its operations, and the results of the regression testing that rely on white-box code coverage may become nonrepresentative.

Input and output information of a service over a test case can be collected without affecting the timing or other QoS constraints of the service. For instance, Mei et al. [20] prioritized regression test suites based on the number of tags encoded in the XML-based messages associated with each actual run of a preceding version of service over the regression test cases. Their techniques, however, are inapplicable if the services do not use XML-based messages for communications. To the best of our knowledge, the location data inputted to real-world LBS-enabled applications are seldom represented in the XML format. Moreover, unlike the techniques to be presented in this paper, their techniques do not interpret the semantics of the data (e.g., using location coordinates in the test cases to compute distances with definitive physical meaning in real world).

Location data provide real-world physical coordinates. Each component of a coordinate is a real number. Because the probability of the same real number appears in a pair of arbitrary coordinate is nearly zero, in general, a realistic test suite is unlikely to contain many identical real number values in its test cases. As such, every such coordinate can be roughly considered as a distinct value. Intuitively, the common idea of many existing general test case prioritization techniques such as the greedy algorithms [9], the search-based algorithms [18], ART [15], the multilevel algorithms [21] on covering “new elements” become inapplicable. For instance, if each test case has the same number of locations in its input, the permutation result of applying the total strategy [9] on such location coverage data set would be degenerated into that of random ordering, which is, on average, ineffective to expose faults in terms of APFD [9]. For the same reason, the effects of the additional strategy [9] and the multilevel algorithms [21] are also degenerated to that of random ordering. Existing search-based algorithms [18] try to optimize the rate of code coverage. Again, as all the values are “distinct,” such an algorithm also behaves like ordering test cases randomly. To the best of our knowledge, these techniques are unaware of the semantics of a set of such real numbers that form a location.

We base our work on two insights. Location estimation algorithms, even the state-of-the-art ones, are inaccurate. Our first insight is that if two locations in close geographic proximity are used as distinct inputs to a POI-enabled application, the application tends to augment the corresponding outputs with similar sets of POI elements than the otherwise. Our second insight is that the POIs in the same output of the application are strongly correlated among them by the application, and they are also correlated to the location data in their corresponding inputs. (Or else, the application seems not location-aware.)

In this paper, we propose a family of black-box test case prioritization techniques that are applicable to test POI-enabled applications. The input-guided techniques aim at maximizing the cumulated amount of uncertainty in the permuted test cases. We propose two location-centric metrics that measure the variance and entropy of the location sequences of each test input. The POI-aware techniques exploit the expected test outputs for test case prioritization, and aims at minimizing the cumulated amount of central tendency of the expected outputs of the permuted test suites to their corresponding inputs. We are not aware of existing test case prioritization techniques that prioritizes test cases based on this dimension. We design two metrics, exploring the central tendency of POIs, and include a metric that measures the amount of POIs covered by individual test cases.

Parts of the results of this paper have been presented in [39] and [40]. We [39] have proposed the above-mentioned five techniques and metrics, and evaluated the techniques using a set of test suites, each containing 1,024 test cases. We [40] have further studied their effectiveness in terms of APFD [9].

In this paper, we further extend the case study reported in [39]. We not only completely reconduct the controlled
experiment and measure the cost-effectiveness through a proposed metric (harmonic mean of rate of fault detection (HMFD) [39]), but also compare the techniques in terms of the overall effectiveness, the stability, the effectiveness under different data sampling rates, and the effectiveness on each major SOA fault class [3].

The case study shows that both the input-guided and POI-aware techniques are more effective than the random ordering in all studied sizes of test suites. Moreover, POI-aware techniques are more effective and more stable than input-guided techniques in the majority of the scenarios. They are also less sensitive to different sampling frequencies on the location data of individual test inputs. We find that, among the studied techniques, cdist is the most effective both in terms of the median and mean HMFDs, the most stable in terms of the standard deviation of HMFD, and the least sensitive to different sampling frequencies on input location sequences. The case study also reveals that no technique always performs the best across all studied SOA fault classes; on the other hand, cdist consistently performs well in detecting faults of all studied SOA fault classes.

The contributions of the paper and the above-mentioned contributions of its preliminary versions are threefold: 1) To the best of our knowledge, we propose the first set of test case prioritization metrics and initialize them as techniques to guide test case prioritization for POI-enabled applications. 2) We present the first case study that validates the feasibility and cost-effectiveness of the initialized techniques. In particular, the case study is the first one that reports the empirical cost-effectiveness of test case prioritization techniques on stateful web services (though our techniques are not restricted to test such web services only), and is the first empirical study that analyzes the cost-effectiveness of regression testing techniques in terms of SOA fault class. 3) We propose HMFD, a new cost-effectiveness metric.

The remainder of the paper is organized as follows: Section 2 presents our prioritization metrics. Sections 3 and 4 describe the setup and the result of the case study, respectively. We review the related work in Section 5, and conclude the work in Section 6.

## 2 OUR MODEL

This section presents our proposed metrics and test prioritization techniques for location-based services.

### 2.1 Preliminaries

Our techniques and metrics rely on concepts related to the geometry and test case prioritization. In this section, we revisit basics terminologies needed by our work.

A location \( l_i = (\text{long}_i, \text{lat}_i) \) is a pair of real numbers representing the longitude and the latitude of the location on the Earth surface. We use real-world coordinates to model every location because our test subjects are LBS-enabled applications and our approach does not rely on other types of information.

We further define the function \( \text{Dist}(x, y) \) to denote the great-circle distance [33] between two locations \( x \) and \( y \) on the Earth surface. Similarly, we define \( \text{Dist}(x, z) \) to denote the shortest distance from location \( x \) to a geodesic polyline \( z \) on the Earth surface.

Moreover, we define the centroid of a set of locations on the Earth surface as a location also on the Earth surface which minimizes the total great-circle distances from it to each location in the set.

A test case \( t = (l_1, l_2, \ldots, l_m) \) is a sequence of locations. We associate each test case \( t \) with a set of POIs denoted by \( P_t \), as a part of the expected output of the test case \( t \).

Modeling a test case as a location sequence is motivated by the observation that a real-world LBS-enabled application usually accepts location data in a rate much higher than the rate to refresh its output. For instance, a smart-phone application may refresh its display once in every 2 seconds, despite that in the same period, tens of location data may have been received by the application.

A test suite \( T = \{t_1, t_2, \ldots, t_n\} \) is a set of \( n \) test cases. The set of all POIs of \( T \) is defined as the union of all \( P_t \) for \( i = 1 \ldots n \).

We adopt the popular test case permutation problem definition from [28], which is as follows: Given: a test suite \( T \), its set of all permutations \( PT \), and a function \( f \) that maps such a permutation to a real number. Problem: Find a test suite \( T' \in PT \) such that \( \forall T'' \in PT, f(T') \geq f(T'') \).

### 2.2 Location-Centric Metrics

This section presents our proposed metrics for test case prioritization. The first two metrics explore randomness in the test input. The next two metrics measure the central tendency of POIs in the expected output of a test case in relation to the central tendency of location data in the corresponding test input.

The last metric investigates the effect of POI coverage in the expected output in relation to the corresponding test input. As such we refer to the first two as input-guided metrics and the remaining three as POI-aware metrics. Table 1 summarizes these metrics.

#### 2.2.1 Sequence Variance (var)

Sequence variance measures the variance of a location sequence of a test case \( t = (l_1, l_2, \ldots, l_m) \). It is defined as follows:

\[
\text{var}(t) \triangleq \frac{1}{|t|} \sum_{i=1, i \neq 0}^{m} \text{Dist}(l_i, \bar{t})^2,
\]

where \( \bar{t} \) is the centroid of all the locations in the test case \( t \).
2.2.2 Polyline Entropy (Entropy)

If we plot the location sequence in a test case \( t \) on the Earth surface, and connect every two consecutive locations in the sequence with a geodesic, we obtain a geodesic polyline, or geodesic curve, with \(|t| - 1\) geodesics and \(|t|\) vertices. Polyline entropy measures the complexity of such a curve. We adapted this metric from euclidean geometry to spherical geometry as follows.

The concept of the entropy of a curve comes from the thermodynamics of curves \([7, 22]\). Consider a finite plane curve \( \Gamma \) of length \( L_\Gamma \). Let \( C_\Gamma \) be the convex hull of \( \Gamma \), and \( C_\Gamma \) be the length of \( C_\Gamma \)'s boundary. Let \( D \) be a random line, and \( P_n \) be the probability for \( D \) to intersect \( \Gamma \) in \( n \) points. The entropy of the curve \([22]\) is as follows:

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

where the maximal entropy \([22]\) can be computed by

\[
P_n = e^{-\beta n} (e^\beta - 1),
\]

with

\[
\beta = \log\left( \frac{2L_\Gamma}{2L_\Gamma - C_\Gamma} \right).
\]

By combining (2)-(4), we obtain the function of the entropy of a plane curve \( \Gamma \) \([8, 22, 29]\):

\[
H(\Gamma) = \log\left( \frac{2L_\Gamma}{C_\Gamma} \right).
\]

We adopt (5) as the polyline entropy of a test case:

\[
\text{entropy}(t) = \log\left( \frac{2L_t}{C_t} \right).
\]

Note that instead of using the original concepts in euclidean geometry, we use their extended versions in the spherical geometry. Hence, \( L_t \) is the length of the geodesic polyline presented by \( t \), \( C_t \) is the boundary length of the convex hull of the curve, and the length of a geodesic is the great-circle distance between the two locations that form it.

2.2.3 Centroid Distance (cdist)

The centroid distance measures the distance from the centroid of a location sequence of a test case \( t \) to the centroid of a set of POIs denoted by \( P \). As the POI information is used in the computation, this metric is a POI-aware metric. We formulate this metrics as follows:

\[
\text{cdist}(t, P) = \text{Dist}(\bar{t}, \bar{p}),
\]

where \( \bar{t} \) is the centroid of all the locations in \( t \), \( \bar{p} \) is the centroid of all POIs in \( P \) associated with the expected outputs of \( t \).

2.2.4 Polyline Distance (pdist)

As mentioned in Section 2.2.2, we consider a test case as a geodesic polyline (curve). Polyline distance measures the mean distance of the set of POIs (denoted by \( P \)) associated with the expected output of the test case \( t \) to the polyline \( z \) of the same test case \( t \). It is defined as follow, where \( P_k \) for \( k = 1, 2, \ldots, |P| \) are elements in \( P \).

\[
pdist(t, P) = \sum_{k=1}^{|P|} \frac{\text{Dist}(P_k, z)}{|P|}.
\]

2.2.5 POI Coverage (pcov)

To compute \( pcov \), we first compute the distances from each POI associated with the expected output of the test case \( t \) to the geodesic polyline \( z \) represented by \( t \). A POI entity is said to be covered by the polyline if the distance from this POI to the polyline is no greater than a value \( \alpha \). Equation (9) defines \( pcov \), where all the symbols carry the same meaning as those in (8):

\[
\text{pcov}(t, P) = \sum_{k=1}^{|P|} \phi(\text{Dist}(p_k, t)),
\]

where

\[
\phi(x) = \begin{cases} 
0, & x > \alpha, \\
1, & x \leq \alpha.
\end{cases}
\]

3 Case Study

This section reports a case study that evaluates the effectiveness of the metrics shown in Table 1. We first present the research questions followed by elaborating on how we initialize these metrics into techniques. To ease our presentation, if a metric is input-guided (POI-aware, respectively), we refer to a technique initialized from it as input-guided (POI-aware, respectively). Finally, we elaborate the setup of the experiment, and analyze the data to answer the research questions.

3.1 Research Questions

To perform regression testing on POI-enabled applications, a technique should be both effective and stable in performance. To facilitate benchmarking, we compare all the techniques with random ordering, which randomly orders the tests in a test suite \([9, 28]\). The primary reason is that our approach does not use any implementation details of the application and does not use any specification. On the other hand, our techniques can be divided into techniques that use the expected output information of test cases and these techniques that do not use such information. It is, therefore, interesting to know whether one type of technique can be more effective and stable than the other. In addition, assuming the availability of complete information is quite restrictive to apply a technique, as signal receivers often filter and down-sample high-speed data stream \([2]\). Hence, a reliable technique should be highly tolerable to different sampling granularities. To study these aspects, we design three research questions RQ1, RQ2, and RQ3 as follows:

RQ1. Are input-guided techniques as effective as POI-aware techniques? To what extent can they be more effective than random ordering? Is there any particular technique always more effective than the other techniques in the case study?

RQ2. Are input-guided techniques and POI-aware techniques more stable than random ordering in terms of
Table 2: Initialized Techniques

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Metric (Sorting order)</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>-</td>
<td>Random ordering</td>
</tr>
<tr>
<td>var</td>
<td>Input-Guided</td>
<td>Eqn. (1): Descending</td>
</tr>
<tr>
<td>entropy</td>
<td>POI-Aware</td>
<td>Eqn. (6): Ascending</td>
</tr>
<tr>
<td>cdist</td>
<td>POI-Aware</td>
<td>Eqn. (7): Ascending</td>
</tr>
<tr>
<td>pdist</td>
<td>POI-Aware</td>
<td>Eqn. (8): Ascending</td>
</tr>
<tr>
<td>pco</td>
<td>POI-Aware</td>
<td>Eqn. (9): Descending</td>
</tr>
</tbody>
</table>

effectiveness? Is the most effective technique identified by RQ1 also the most stable technique?

RQ3. To what extent can a technique tolerate downsampling on the input location sequences without significantly compromising the effectiveness of the technique? How well is the most effective technique or the most stable technique in this tradeoff aspect?

A technique that performs well in all above three aspects but fails to expose some important classes of fault is not practical enough. We design RQ4 to empirically study the performance of each technique at the SOA fault class level (see Section 3.6 for the classification).

RQ4. Are input-guided techniques and POI-aware techniques effective (ineffective, respectively) to detect faults in each studied SOA fault class? For a given SOA fault class, what is the relative effectiveness of input-guided techniques in relation to POI-aware techniques? Is there any particularly effective technique across all SOA fault classes in the study? Is this technique, if exists, also the same technique identified in RQ1, RQ2, or RQ3?

3.2 Initialized Techniques

To investigate the metrics proposed in Section 2.2, we apply each metric on each test case in a test suite to collect the metric value of each test case, and then adopt the total strategy [28], which essentially prioritizes the test cases according to their corresponding metric values. This simplicity allows other researchers to use them as baselines to benchmark their own studies.

In the rest of this section, we elaborate our rationales to initialize each metric summarized in Table 1 into a technique that sorts test cases in certain order. To simplify our presentation, we use the metric name to stand for the initialized technique in this case study. Note that all techniques resolve tie cases randomly. Table 2 summarizes the initialized techniques as well as the random ordering [9], [28] for the case study. The basic ideas of the five initialized techniques are third-party services. We do not mutate their codes to produce faulty versions. The function of the Android client is straightforward, which merely encapsulates and passes the GPS data to the server and displays the results with Google Maps. We exclude these three services from the case study. Upon receiving the data, this service interacts with a remote server. We use this service as the subject of our case study. We use a stateful POI-enabled application CityGuide [39] as the case study application, and Fig. 1 shows its system architecture. CityGuide includes a client collecting the GPS data and displaying a Google Maps instance on the Android platform for mobile phones. It sends these GPS data and other phone settings to a service located on a remote server. We use this service as the subject of our case study. Upon receiving the data, this service interacts with a case-based reasoning engine [1], constructs a message embedded with POI data to the client that uses Google Maps to visualize the POIs.

Both Google Maps and the case-based reasoning engine are third-party services. We do not mutate their codes to produce faulty versions. The function of the Android client is straightforward, which merely encapsulates and passes the GPS data to the server and displays the results with Google Maps. We exclude these three services from the subject application in the study. The case-based reasoning engine is not a merely query processing tool. Rather, it is developed with a set of similarity metrics and is initialized with “cases” for similarity-based matching. For each recommended POI, the user can optionally confirm whether the POI is relevant to what user wants to know and update his/her personal preferences. Then, the case-based engine will add this scenario as a case in the case database.

We configured a mutation tool (MuClipse) and applied all the mutation operators described in [30] to the subject service to produce faulty versions. Each generated mutant was considered as a faulty version of our subject service.
Similar to the previous experiments [9], [21], we deemed the original version of the subject as the \textit{golden version}.

The mutation tool generated many faulty mutants. We followed the existing test case prioritization experimental procedure to exclude all mutants with faults that either cannot be detected by any test case or can be detected more than 10 percent of all test cases [9]. After this filtering process, 35 mutants are remained, and we used all of them for data analysis.

Table 3 summarizes the descriptive statistics of the subject service. Our subject service consisted of 3,289 noncomment lines of code. In terms of the experimentation for service testing, the experiment presented in [21] used a suite of eight BPEL programs with 60 faults. The combined size of these BPEL programs was smaller than the size of our subject service.

We chose to conduct a case study rather than a controlled experiment that involves many services to validate the proposed techniques, primarily because each service required its unique working environment, and we had exercised our best efforts to evaluate the proposed techniques under our limited human resources. For instance, in our case study, the service was not a simple standalone program. Rather, it was part of a larger application, which consisted of a database server, a case-based reasoning engine, an Android client service and phone emulator, and an Apache web server. The efforts to configure the application, to restore the contents of database and web server, to cleanse test garbage, and to run the application to support the empirical validation were nontrivial.

To set up the application for the experiment, we collected 104 real hotels on the Hong Kong Island (which has a population of 1.3 million) of Hong Kong as POIs and then initialized the case base engine with these POIs. In particular, to avoid bias, we included all the hotels in the Wan Chai and Causeway Bay districts of the Hong Kong Island. Sample POIs visualized by Google Map are shown in Fig. 2. It is worth noting that each hotel appeared at least once in the expected results of our test cases.

We used the default similarity metrics of the case-based engine. The engine jColibra2 is downloadable from \url{http://gaia.fdi.ucm.es/projects/jcolibri/}. We configured the engine to use MySQL v5.1.39 to keep its data.

We generated a test pool of 2,000 random test cases, each of which was a sequence of legitimate GPS locations on the Hong Kong Island. When generating the test pool, we made assumptions on the behaviors of the people (both walking and driving) according to the GPS track data obtained from public sources like OpenStreetMap [25]. To further ensure the realism of the test pool, one of the authors (Zhai) sampled a small amount of these location sequences, and examined the sequences one by one to judge their validity by our living experience on the Hong Kong Island. All test suites used in the experiment were selected from this test pool. The descriptive statistics of the generated test cases are shown in Table 4.

Fig. 3 illustrates the fault coverage of our test pool. The $x$-axis specifies the number of faults, and the $y$-axis specifies the number of test cases that can detect the number of faults given by $x$-axis. Note that the $y$-axis uses a log scale. There are 1,138 test cases that cannot detect any fault, and no test case can detect more than 23 faults.

Fig. 4 further shows the distribution of all 2,000 test cases in terms of the ratio of all POIs covered by each test case. We found that 705 test cases cover not more than 10 percent of all POIs each in terms of the $pcov$ measure by setting $\alpha$ to be 1 kilometer. As expected, the data does not follow obvious distribution patterns because the POIs are real hotels rather than artificial data points.

To know whether a failure in a faulty version of the subject service was detected by a test case, our tool used the last response message by the faulty version as the actual output of the faulty version, and compares it with that of the golden version. If there is any difference (except the

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
<th>Faulty Versions</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CityGuide</td>
<td>Hotel recommendation</td>
<td>35</td>
<td>3,289</td>
</tr>
</tbody>
</table>

Table 3 Subject Service Description

<table>
<thead>
<tr>
<th>Number</th>
<th>Length</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000</td>
<td>32</td>
<td>114°9′E</td>
<td>114°13′E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22°15′N</td>
<td>22°18′N</td>
</tr>
</tbody>
</table>

Table 4 Statistics of Generated Test Cases

Fig. 2. Sample POIs used in the case study. They are hotels in Wan Chai and Causeway Bay of Hong Kong Island.

Fig. 3. The distribution of test cases that can detect a given number of test cases.

Fig. 4. The distribution of test cases that cover different percentages of all hotels as POIs.
3.4 Experimental Environment

We conducted the experiment and data analysis on a Dell Power Edge 1950 server. It ran a Solaris UNIX operating system, and was equipped with two Xeon 5355 processors (each is quad-core with 2.66 GHz) and 8-GB physical memory.

3.5 Effectiveness Metric

To measure how quickly a prioritized test suite can detect faults, we propose to use the harmonic mean (HM), which is independent from the size of a given test suite. The HM is a standard statistical tool to combine a set of numbers, each representing a rate, into an average value [4]. Moreover, one merit of the HM over the arithmetic mean is that the HM is less affected by the extreme values in the given set of numbers.

The proposed metric is defined as follows: Let $T$ be a test suite consisting of $n$ test cases and $F$ be a set of $m$ faults revealed by $T$. Let $T_{Fi}$ be the first test case in a reordered test suite $S$ of $T$ that reveals the fault $i$. The harmonic mean of the rate of fault detection (HMFD) of $S$ is defined as follows:

$$HMFD = \frac{m}{\frac{1}{TF_1} + \frac{1}{TF_2} + \cdots + \frac{1}{TF_m}}.$$ 

We use HMFD to measure the effectiveness of a technique. Note that a lower HMFD value indicates a more effective result. We also use the standard deviation of HMFD achieved by a technique to measure the stability of the technique.

Let us use an example to clarify the metric further. Suppose that a program has three faults and we are given with a test suite with five test cases. Suppose further that the first test cases in a reordered test suite $S$ detect the fault $i$ are the first, second, and third test case, respectively. The HMFD value for $S$ is, therefore, computed as $3/(1/1 + 1/2 + 1/3)$, which is 1.64.

Similarly, suppose that there is another test suite that contains 10 test cases and can detect the last two faults in the same program, but misses to detect the first fault. Suppose further that the first test cases in a reordered test suite $S'$ can detect the two faults are the second test case and the third test case, respectively. The HMFD value for $S'$ is $2/(1/2 + 1/3) = 2.4$, which is higher than 1.64.

Many existing experiments use the weighted average percentage of fault detection (APFD) [9] as the metric to evaluate the effectiveness of a prioritization technique to expose faults. For the above two scenarios, their APFD values are 0.70 and 0.88 (a higher value is more favorable for APFD), respectively. Intuitively, the first reordered test suite should be more effective than the second one, since it actually detects one more fault by its first test case, but the above APFD values hints the otherwise. More discussion on APFD can be found in Section 4.6.

3.6 Classification of SOA Faults

We classified each of the 35 faults of the subject service into a SOA fault class using the top-level fault taxonomy for service-oriented architecture proposed in [3]. To determine the fault class of a fault, we conducted code inspection to assess the kind of fault exhibited by each mutant. For instance, a discovery fault [3] may not be manifested into a failure until the service is executed. In such a case, this fault may also naturally be classified as an execution fault [3]. However, since the fault is located in the code region for service discovery, we did not count this fault as an execution fault. Rather, we counted it as a discovery fault. We used this strategy to classify each of 35 faults to exactly one fault class. The categories we used to classify faults are described as follows.

The first fault class is the publishing fault [3]. During service publication, a service is deployed on a server and its service description is made public so that the service can be discovered and invoked. A failure of a publishing fault may occur if the description of the service is incorrect or the deployment is problematic. For instance, the service and the hosting server may be incompatible. In the case study, publishing faults are serious and cause frequent service failures. Because of the above-mentioned filtering procedure used by the experiment, we found no fault of this kind.

The second fault class is the discovery fault [3]. Failures during service discovery may occur at invocation to initialize a service discovery or when receiving notification of candidate services from the environment of the service. For instance, a fault of this kind may result in a failure discovering no service even though the required services are available. Discovering a wrong service may be hard to detect, since this fault may manifest into a failure only when the service is executed. The case study had seven discovery faults.

The third fault class is the composition fault [3]. CityGuide is a kind of typical service composition that requires specific interaction sequences among services. There may be many reasons that result in faulty service compositions. For example, incompatible services may be erroneously composed together. Some composition faults can also be found in the contract (interface) between services in a composition. Four composition faults were identified in the case study.

The fourth fault class is the binding fault [3]. Even though there is no fault in service discovery and composition, an application may still be mistakenly bound to a wrong service or fail to bind to any service. The case study had six binding faults.

The last fault class is the execution fault [3]. Apart from the SOA-relevant components that support service publishing, discovery, composition, and binding, we consider faults in all other components of a service as execution faults. This type of faults has been well studied for programs [8] that are not categorized as services. We had 18 faults of this type.

Fig. 5 shows the fault detection ability of the test pool on each SOA fault class. The $y$-axis lists the four SOA fault classes. The $x$-axis shows the number of test cases in our test pool that can detect at least one fault in the given SOA fault class by the $y$-axis.

3.7 Experimental Procedure

We developed a tool that implemented each technique and the random ordering. Because the size of a regression test
suite may affect the effectiveness of the techniques, to study RQ1 and RQ2, we systematically varied the size of a regression test suite to be 128, 256, 512, and 1,024. We used these sizes by referencing the sizes of the test suite of software of similar scale. For instance, the test suite sizes for gzip, sed, flex, and gzip in existing test experiments (e.g., [15]) are 217, 370, 567, and 809, respectively. For each test suite size, the tool constructed 50 test suites by randomly selecting test cases from the test pool. Note that in order for a fair comparison among test suites of the same size, all used test suite are selected such that they are able to detect all 35 faults.

To study RQ3, for each test suite, the tool further sampled the location data from selective index positions of the GPS sequence of each test case. The tool systematically extracted location data from every $f$ index positions (starting from the first position of the sequence), where $f = 1$ (i.e., full sampling), 2, 3, 4, 5, 6, 8, and 10. Owing to each GPS sequence included in each test case containing 32 GPS locations, the lengths of the sampled location data are set to be the floor values of $32/f$, which are 32, 16, 10, 8, 6, 5, 4, and 3, respectively. For instance, at $f = 2$, the tool extracted the location data from each GPS sequence $s$ at positions $s[1], s[3], s[5], ..., s[31]$, where $s[i]$ stands for the location data kept at the index position $i$ of $s$. We refer to $f$ as the sampling frequency.

For each test suite and each sampling frequency, the tool ran each technique to generate a permuted test suite. The tool applied each test case to each faulty version, and compared the corresponding output with that of the golden version. It reported the HMFD value achieved by the permuted test suite. The tool also collected the POIs of a test version. It reported the HMFD value achieved by the compared the corresponding output with that of the golden technique. Out of the 24 comparisons, 20 show that (that is, two) between an input-guided technique and a POI-aware technique also show large difference.

For each technique, we calculated the HMFD values on all faulty versions, and drew four box-and-whisker plots for test suites of size 128, 256, 512, and 1,024, as shown in Fig. 6. For each box-and-whisker plot, the $x$-axis represents prioritization techniques and the $y$-axis represents the HMFD values achieved by a technique at $f = 1$ (i.e., full sampling). The horizontal lines in each box indicate the lower quartile, median, and upper quartile values. We also used MATLAB to perform the analysis of variances (ANOVA) followed by a multiple-comparison procedure [15, 18] to find those techniques whose means are different significantly from those of others at the 5 percent significance level. The $p$-values for the ANOVA tests for the suite size of 128, 256, 512, and 1,024 are $2.75 \times 10^{-14}$, 0, 0, and 0, respectively, which show that there are significant differences in each case. The multiple-comparison results are shown on the right-hand side of each box-and-whisker plot in Fig. 6. The multiple-comparison procedure shows the distribution of HMFD value of each technique as a horizontal line with a dot in the center, which denotes the mean value with a confidence interval at the 95 percent confidence level.

From the subfigures (1), (3), (5), and (7) of Fig. 6, we observe that, in terms of the median, the POI-aware techniques are more effective than the input-guided techniques. For instance, the ratio of the median HMFD of $cdist$ to that of entropy in the subfigure (1) is 45.11 percent, which is significant. The corresponding ratios for each other pair of an input-guided technique and a POI-aware technique also show large difference.

The multiple comparison results from subfigures (2), (4), (6), and (8) of Fig. 6 are consistent with the above observation. Each plot shows six comparisons (three times two) between an input-guided technique and a POI-aware technique. Out of the 24 comparisons, 20 show that (that is, except the bars between entropy and $pdist$ and between entropy and $pcov$ in subfigures (4) and (6)), the mean HMFD values of POI-aware techniques are smaller than those of input-guided techniques at the 5 percent significance level.
Moreover, we observe that all proposed techniques are superior to random ordering in most of the cases. The subfigures (2), (4), (6), and (8) of Fig. 6 also show that the difference between random ordering and var increases as the size of a test suite increases. Other techniques have less obvious trends. Last, but not the least, we observe that cdist is the most effective technique among the studied techniques both in terms of median and mean HMFD values.

### 4.2 Answering RQ2

To know whether a technique is stable in terms of HMFD, for each sampling frequency of each test suite size, we computed the standard deviation of the HMFD values achieved by each permuted test suite. The result is shown in Fig. 7, where the x-axis represents the techniques and the y-axis represents the standard deviation of the HMFD value achieved by each technique on all test suites. Moreover, we conducted ANOVA via MATLAB, and the p-value of the ANOVA test is 0. Based on the test result, we further conducted a follow-up multiple mean comparison analysis as shown in Fig. 8.

We observe that the standard deviation of HMFD value achieved by the random ordering is larger than that of achieved by each input-guided or POI-aware technique both in terms of median and mean HMFD. Furthermore, we observe that input-guided techniques exhibit much larger standard deviations than POI-aware techniques. For instance, from Fig. 8, we find that the mean standard deviations of the input-guided techniques are almost two times as those of the POI-aware techniques. For example, random ordering achieves both higher mean (5.587) and higher standard deviations (3.427) of changes in effectiveness than all three POI-aware techniques.

Moreover, we find that on average, input-guided techniques are much more sensitive to different ratios of location samples extracted from test cases than POI-aware techniques.

### 4.3 Answering RQ3

In this section, we analyze the data to study the influence of sampling frequencies on the effectiveness of prioritization techniques.

For each selected suite size, we analyze how the mean HMFD values achieved by the technique change as the sampling frequency $f$ increases. The result is shown in Fig. 9, where x-axis represents the values of $f$ used to extract location data. The y-axis shows the mean HMFD values achieved by the techniques on the test suites downsampled for a given $f$ value.

As expected, the effectiveness of random ordering is unaffected by the sampling process, because the random ordering does not use any information specific to a location sequence.

Across all these techniques, when the sampling frequency increases, the mean HMFD values of input-guided techniques deteriorate faster than POI-aware techniques. We also observe that this deterioration is more significant on larger test suites than on smaller test suites.

Moreover, we find that input-guided techniques are much more sensitive to different ratios of location samples extracted from test cases than POI-aware techniques.
techniques. In particular, \textit{cdist}, which is the most effective and stable technique identified in Sections 4.1 and 4.2, is also the least sensitive technique for each suite size.

4.4 Answering RQ4
In this section, we empirically examine the effectiveness of each proposed technique and the random ordering on SOA fault classes. For each technique, we partitioned the set of 35 faults into the SOA fault classes, and computed the HMFD values at $f = 1$ (full sampling). The results are shown in Fig. 10. Each bar in the plots illustrates the effectiveness of one technique for a given class of fault.

The interpretation of $x$- and $y$-axes of each plot is similar to those of Fig. 6. Each bar shows three quartiles that divide the sorted data into four equal parts.

Subfigure (1) of Fig. 10 shows the effectiveness of techniques for discovery faults in detecting composition faults. We observe that all techniques are more effective than random ordering. However, POI-aware techniques are not in general more effective than input-guided techniques. It is understandable because discovery fault is related to how a service discovers other services, and it is not quite related to the semantics of the service to compute an output for a test input. We also find that \textit{pdist} is the most effective technique.

From subfigure (2) of Fig. 10, all proposed techniques are more effective than the random ordering. However, the differences between input-guided techniques and POI-aware techniques are smaller than the corresponding differences shown in subfigure (1). For instance, \textit{entropy} is more effective than \textit{pdist} in terms of mean, although the difference is not statistically significant according to the multiple comparison analysis.

For the binding faults and execution faults shown in subfigures (3) and (4) of Fig. 10, respectively, two POI-aware techniques (\textit{cdist} and \textit{pdist}) outperform all input-guided techniques, which in turn outperforms the random ordering.

From the result, we have seen that the proposed POI-aware techniques are observably more effective than input-guided techniques and the random ordering. However, to reveal discovery and composition faults, the difference may not be significant statistically.

In general, we find that both input-guided and POI-aware techniques are effective to detect each studied type of fault as indicated by the large difference (in terms of mean HMFD) between each of them and random ordering. Moreover, we find that the effectiveness of these techniques is similar, although POI-aware techniques tend to be more effective in the detection of the binding and execution faults. However, there is no obviously most effective technique that excels at all studied fault classes, and \textit{cdist} can be less effective than some other techniques to detect faults of the discovery and composition fault classes. Fortunately, even though \textit{cdist} is not always the most effective technique for every fault class, it is either the most effective technique or the second most effective technique among all the studied techniques.

4.5 Result Summary
The case study shows that the proposed POI-aware and input-guided techniques are more effective than the random ordering for all studied sizes of test suites in terms of HMFD. Moreover, POI-aware techniques are less sensitive to change in sampling frequencies to extract location data from individual test cases and change in test suite sizes than input-guided techniques. In other words, the performance of POI-aware techniques can be more stable.

Concerning the rate of fault detection, we find from the case study that the POI-aware techniques tend to be more effective than input-guided techniques in terms of median and mean HMFDs. In general, technique \textit{cdist} is the most effective and stable technique. It performs either as the best or the second best technique in each studied aspect. To detect SOA discovery fault, \textit{pdist} can be better. In general, the case study shows that POI information can be valuable

<table>
<thead>
<tr>
<th>Technique</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
<th>Mean</th>
<th>Std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{var}</td>
<td>2.2018</td>
<td>1.9426</td>
<td>5.9714</td>
<td>8.9106</td>
<td>5.5874</td>
<td>3.4271</td>
</tr>
<tr>
<td>\textit{entropy}</td>
<td>1.5045</td>
<td>5.1243</td>
<td>6.2808</td>
<td>6.9275</td>
<td>5.3529</td>
<td>2.2739</td>
</tr>
<tr>
<td>\textit{cdist}</td>
<td>0.1502</td>
<td>0.4322</td>
<td>0.1012</td>
<td>0.1018</td>
<td>0.1774</td>
<td>0.1440</td>
</tr>
<tr>
<td>\textit{pdist}</td>
<td>1.5542</td>
<td>2.1092</td>
<td>3.0180</td>
<td>3.8011</td>
<td>2.8567</td>
<td>1.0081</td>
</tr>
<tr>
<td>\textit{pcov}</td>
<td>1.2125</td>
<td>1.7891</td>
<td>0.6900</td>
<td>1.1592</td>
<td>1.2020</td>
<td>0.3907</td>
</tr>
</tbody>
</table>

Table 5: Difference between Mean HMFD Value at $f = 1$ and That at $f = 10$ of Each Technique

Fig. 10. Performance of techniques on different SOA fault classes.
to guide test case prioritization for the regression testing of LBS. We finally summarize our results for each research question in Table 6.

### 4.6 Threats to Validity

**Threats to external validity.** In this study, we used a service in a POI-enabled application to evaluate the techniques initialized from the proposed metrics. It is possible for the application to use other reasoning engines like rule-based, ontology-based, or simply geospatial database. The subject is implemented in the Java language. Since our techniques require black-box information only, they are applicable to POI-enabled service implemented in other programming languages. However, the corresponding set of faults may be different, which may produce different HMFD values even using the same set of permuted test cases. Our case study was also limited by the POI data set. We set up them using the real-life data available in Hong Kong. Using other data sets may produce different results. Hong Kong attracted over 29 million tourists in 2009. These tourists may need finding hotels for their accommodations. We believe that the hotel information that the case study captured simulates realism. Another threat relates to the current initialization of the proposed metrics into sort-based techniques. Other initializations are possible. The interpretation of the current results to other types of initialization should be done carefully. A more comprehensive study on comparison with other black-box techniques, especially those nonsort-based techniques, may strengthen the validity of our work.

**Threats to internal validity.** A threat to internal validity is the correctness of our tools. We used Java (Eclipse) and MATLAB for tool implementation, test suite prioritization, and results comparison and plot generation. To reduce the chance of faults affecting our results, we had reviewed and tested our tool with some test cases and manually verified the results. We used the last response message returned by the subject over a test case to compare with that of the golden version to determine whether there is a failure. Obviously, using other types of oracle checks may produce different results. We chose this setting because in test experimentation, researchers popularly used the output (and ignore the intermediate data) of the program to stand for the actual output of the program. In the case study, we followed this strategy to set up the oracle checking mechanism. Our experiment used all the hotels located in the Wan Chai and Causeway Bay districts to simulate the realism for the CityGuide application. Another threat is the classification of the seeded faults into the SOA fault class. We used the code locations of the faults to identify the fault classes for these faults. We have reinspected the code against the classification results to validate the classification correctness.

**Threats to construct validity.** We measured the rate of fault detection by HMFD. As explained in the setup of the case study, HMFD is the harmonic mean of a set of numbers, each of which stands for the number of test cases needed to be executed to reveal the presence of a fault in its first time. Because such a number stands for a “rate” rather than merely a number, according to mathematical textbooks, the use of the harmonic mean of these rates can be more representative than taking the arithmetic mean of these “rates.” Using other metrics may give different results.

For instance, APFD has been widely used in previous experiments [9], [15], [21], [39]. APFD measures the weighted average of the percentage of faults detected over the life of the suite. APFD is, however, a variant of the arithmetic mean of these rates. Furthermore, there are some issues around APFD to motivate us to use HMFD instead.

Let $T$ be a test suite containing $n$ test cases, and let $f$ be a set of $m$ faults revealed by $T$. Let $TF_i$ be the index of the first test case in the reordered test suite $S$ of $T$ that reveals fault $i$. The APFD value of $S$ is given by the following equation [10]:

$$ APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n}. $$

Apart from the issues that we have presented in Section 3.5, readers may easily observe another limitation of the APFD measure, which is as follows.

Equation (10) shows that the APFD value is affected by the test suite size $n$. How does the test suite size affect the APFD value? We could estimate the trend as follows.

Suppose that there is a sufficiently large test pool $T_p$ and $m$ faults. The failure rate of fault $i$ with respect to $T_p$ is $R_i$. For a test suite $T$ of size $n$ extracted from $T_p$, we compute a permutation of $T$. For each fault $i$, we use the notation $A_{ij}$ to denote the event that the $j$th test case kills fault $i$. Because $T_p$
is sufficiently large, we have \( \Pr(A_{ij}) = R_i \). Hence, the expectation of \( TF_i \) can be expressed as follows:

\[
E(TF_i) = \sum_{j=1}^{n} \left( j \cdot \Pr(A_{ij}) \cdot \prod_{k=1}^{j-1} \Pr(A_{ik}) \right) = \sum_{j=1}^{n} \left( j \cdot R_i \cdot (1 - R_i)^{j-1} \right) \leq \sum_{j=1}^{n} \left( j \cdot R_i \cdot (1 - R_i)^{j-1} \right) = \frac{R_i}{1 - R_i} \sum_{j=0}^{\infty} (j \cdot (1 - R_i)^j) = \frac{R_i}{1 - R_i} \frac{1 - R_i}{R_i^2} = \frac{1}{R_i}.
\]

Let \( R_{\text{min}} \) denote the minimum among \( R_1, R_2, \ldots, R_m \). We have

\[
E(TF_i) \leq \frac{1}{R_{\text{min}}}. \]

The expected APFD of the permuted test suite is

\[
E(\text{APFD}) = 1 - \frac{\sum_{i=1}^{n} E(TF_i)}{nm} + \frac{1}{2n} \geq 1 - \frac{1}{n R_{\text{min}}} + \frac{1}{2n} = 1 - \left( \frac{1}{R_{\text{min}}} - \frac{1}{2} \right) \frac{1}{n}.
\]

For a particular test pool and a set of faults, the \( R_{\text{min}} \) can be regarded as a constant and \( R_{\text{min}} < 2 \). Hence, from (12), we know that

\[
E(\text{APFD}) = \Omega \left( 1 - \frac{1}{n} \right). \tag{13}
\]

On the other hand, we know that each \( TF_i \) must be greater than or equal to one. Hence, we have

\[
E(\text{APFD}) \leq 1 - \frac{1}{n} + \frac{1}{2n} = 1 - \frac{1}{2n} = O \left( 1 - \frac{1}{n} \right). \tag{14}
\]

Combining (13) and (14), we have

\[
E(\text{APFD}) = \Theta \left( 1 - \frac{1}{n} \right).
\]

We observe that the APFD measure grows with the increasing size of the test suite, so APFD may not be meaningful either to compare prioritization effectiveness of different techniques if the test suites are of different sizes or to aggregate the results from a set of test suites of different sizes, such as the result shown in Figs. 7 and 8.

Indeed, we have evaluated the proposed techniques by the APFD as well. Part of the results is published in [40] which are consistent with the conclusion presented in this paper. For instance, Fig. 4 of Zhai and Chan [40] evaluated the effectiveness of proposed techniques in terms of APFD. We observe that the relative effectiveness among techniques in that figure is consistent with our result of RQ1 in this paper. Moreover, readers may get an impression on the variances of different techniques from Fig. 4, which are consistent with our results for RQ2 in this paper. For RQ3, since the variances of the POI-aware techniques are shown to be lower in terms of APFD in Zhai and Chan [40], we conjecture that their performances are less sensitive to the sampling frequency, and yet more experiments are required to confirm the conjecture. The result for RQ4 if APFD is used as the measurement also requires further investigation.

5 RELATED WORK

In previous studies, many test case prioritization techniques were proposed. Many of which were coverage-based for white-box testing.

Li et al. [18] proposed a metaheuristics approach to generating permuted test suites. Jiang et al. [15] proposed a family of adaptive random test case prioritization techniques that tried to spread the test cases across the code space to increase the rate of fault detection. Elbaum et al. [9] studied the effect of the total and additional code coverage strategies to reorder test suites. Our approach does not rely on code coverage.

More importantly, coverage-based techniques rely on the presence of redundancy on coverage requirements (e.g., the same statement) among test cases to differentiate or prioritize these test cases. For a set of location values (test cases) that each of them is a distinct real number (representing locations), it is unlikely to have multiple identical real numbers in general. The consequence is that such coverage-based techniques would behave like random ordering when prioritizing these test cases because there is no or very little coverage redundancy. The ART techniques presented in [15] used the Jaccard distance to compute the difference between two test cases. One may substitute the Jaccard distance by our metrics. The use of the proposed metrics for metaheuristics algorithms is also possible.

Some studies explore alternate coverage dimensions. Walcott et al. [35] and Zhang et al. [41] proposed time-aware prioritization techniques to permute test cases under the given time constraints. Srivastava and Thiagarajan [31] proposed to compare different program versions at the binary code level to find their differences and then prioritize test cases to cover the modified parts of the program maximally.

There is also a handful amount of work focusing on prioritization for black-box testing. For example, Qu et al. [26] proposed a prioritization approach for black-box testing. The proposed approach adjusts the order of the test cases dynamically during the execution of the test suite with respect to the relationship among the test cases. However, the relationship is obtained from history and can be invalid on the evolved software. A comprehensive survey of selective test case prioritization techniques can be found in [38].

Researchers also investigated the problem of regression testing of service-oriented applications. In [21], Mei et al. proposed a hierarchy of prioritization techniques for the regression testing of services by considering different levels of business process, XPath, and WSDL information as the methods to resolve tie cases faced by prioritization. In [21], they also studied the problem of black-box test case
prioritization of services based on the coverage information of WSDL tags. Different from their work that explored the XML messages structure exchanged between services to guide test case prioritization, our techniques used data contents as the sources for prioritization. In [12], Hou et al. observed that service invocations may incur costs, and proposed to impose quotas on the number of allowable invocations for some services. They further use this heuristics to develop test case prioritization techniques.

Furthermore, we can classify the measures used by these prioritization techniques to evaluate test cases as absolute measures or relative measures. Absolute measures summarize the characteristic of a particular test case while the relative measures describe the relationship among a group of test cases. When the values of absolute measures are available, sorting is a widely used approach to permute the test cases. An example use of absolute measures is in the total statement coverage technique [28], where each test case is summarized by the number of statements it covers. On the contrary, an example of relative measures is in the additional statement coverage technique [28], where each test case is measured by its relationship with some existing test cases. The location-centric metrics proposed in this paper are absolute measures which are independent of the choice of test cases. The computation complexity and maintenance efforts are both lower for absolute measure. In our future work, potential relative measures will be investigated and evaluated for location-based services.

Researchers also study methods to integrate test case prioritization with the other techniques. For instance, Wong et al. [37] proposed to combine test suite minimization and test case prioritization to select cases based on their cost per additional coverage. Jiang et al. [16] studied the effect of test adequacy [42] on test case prioritization and statistical fault localization techniques.

6 CONCLUDING REMARKS

Location-based services provide context-sensitive functions based on location information. They in general have runtime constraints to be satisfied and some subservices are not within the control of the service providers. In practice, after a service has been deployed, it may modify existing features, add new features, or correct faults. To assure the modifications not affecting the consumers of the service, service providers may perform regression testing to detect potential anomaly on the services.

Regression testing can be an efficient approach to identifying service problems. However, white-box techniques require extensively code-coverage statistics, which may significantly compromise the runtime behavior of location-based services. On the other hand, specification-based techniques may not be widely applicable because many specifications are often incomplete or outdated. The most popular black-box approach to arrange test cases remains the random ordering, which, however, can be ineffective to detect service anomalies efficiently.

Locations and points of interest are correlated by their location proximity via the services. Moreover, many location-based applications use estimated locations and treat similar locations homogeneously. They provide domain-specific heuristics to guide testing techniques to prioritize test cases.

In this paper, we have proposed a set of location-centric metrics and black-box input-guided as well as POI-aware test case prioritization techniques. We have also reported a quantitative case study to evaluate the effectiveness of our metrics and the initialized techniques using a stateful service application as the subject. The empirical result of the case study has shown that the POI-aware techniques are more stable and more effective in terms of HMFD than random ordering on various test suite sizes. They also outperform input-guided techniques in most of the studied scenarios. We have also found that cdist, one of the POI-aware test case prioritization techniques, can be effective, stable, and least sensitive to the change in downsampling on test inputs (relative to other studied techniques). This technique is also either the most effective or the second most effective technique to detect faults in the four SOA fault classes in the case study both in terms of the mean and median HMFD values.

The case study has also revealed that the optimal technique across all the studied SOA fault classes has not been identified, which indicates that more research should be conducted in this area. In particular, in the case study, faults of the discovery fault class are more difficult to be detected effectively and efficiently. The technique cdist cannot perform very well on this, whereas an input-guided technique, entropy, can be better. The underlying reason is still unclear to us.

Location-based services can be considered as context-aware systems. One may consider extending the proposed metrics and techniques to cover these systems.

One may deem the technique pcov as a variant of the total prioritization strategy reported in earlier work. However, the case study in this paper has shown that the technique is less effective, less stable, and sometimes is worse than input-guided techniques. Moreover, in a majority of studied cases, pcov is the worst technique among the three POI-aware techniques.

To sum up, the proposed POI-aware metrics can be promising in the regression testing of POI-enabled services. In the case study, their initialized techniques can outperform those techniques that do not use any POI information like input-guided techniques and the random ordering in terms of effectiveness and stability. In the future, it is interesting to further explore different prioritization mechanisms that test dynamic service composition, which may require further extension of the metrics to deal with the dynamic evolution nature of dynamic service compositions.

ACKNOWLEDGMENTS

This work was supported in part by the General Research Fund of Research Grants Council of Hong Kong (project nos. 111410 and 717811), the National Natural Science Foundation of China (project no. 61202077), and the CCF-Tencent Open Research Fund (project no. CCF-TencentAGR20130111). Bo Jiang is the correspondence author.

REFERENCES


I. Todhunter, Spherical Trigonometry: For the Use of Colleges and Schools, pp. 1-7, Macmillan, 1911.


Ke Zhai received the bachelor’s and master’s degrees from the Software College of North-eastern University, Shenyang, China, in 2007 and 2009, respectively. He is currently a technology analyst Goldman Sachs and working toward the PhD degree at the University of Hong Kong. His research interests include software testing and debugging, concurrent system analysis and verification, and engineering issues in services computing. He is a student member of the IEEE.

Bo Jiang received the PhD degree from the Department of Computer Science, University of Hong Kong, in 2011. He is an assistant professor in the School of Computer Science and Engineering, Beihang University, China. His research interests include domain-specific testing, debugging, verification, and simulation techniques on embedded, avionics, mobile, and service computing applications. He is a member of the IEEE.

W.K. Chan received all his degrees from the University of Hong Kong. He is an assistant professor in the Department of Computer Science, City University of Hong Kong. He is currently an editorial board member of the Journal of Systems and Software. His latest research interests include address software analysis and testing challenges in large-scale concurrent software. He is a member of the IEEE.