Point-of-Interest Aware Test Case Prioritization: Methods and Experiments*

Ke Zhai
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
kzhai@cs.hku.hk

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

***Abstract*—Location based services personalize their behaviors based on location data. When data kept by a service have evolved or the code has been modified, regression testing can be employed to assure the quality of services. Frequent data update however may lead to frequent regression testing and any faulty implementation of a service may affect many service consumers. Proper test case prioritization helps reveal service problems efficiently. In this paper, we review a set of point-of-interest (POI) aware test case prioritization techniques and report an experiment on such techniques. The empirical results show that these POI-aware techniques are more effective than random ordering and input-guided test case prioritization in terms of APFD. Furthermore, their effectiveness is observed to be quite stable over different sizes of the test suite.**

Keywords-test case prioritization; location-based service

I. INTRODUCTION

Location-based service (LBS) is indispensible to our modern life. Nowadays, with a GPS navigator or GPS-enabled mobile phone as a terminal service, we can enjoy the convenience brought to us by rich location-based applications.

One kind of such services is the location-aware information service [24] where the digital distribution of information is based on device location, time specificity, and user behavior. For example, the *Google Places Directory* [9] enables us to find nearby hotels and restaurants of interest. Such a specific location (e.g. hotel) is popularly known as a *point of interest* (POI) [11][22]. On the other hands, POIs are not limited to be static objects in terms of time and space. They can be dynamic as well. For instance, the set of POIs relevant to a user may change dynamically as in the social network application *Loopt* [10], where a user receives notifications whenever the friends of the user are in his/her physical proximity of the user. In such an application, the locations of these friends are the POIs.

However, the location-based services may evolve frequently even after a formal release. For example, the Google Places Directory may update its database to enable its users to search recently added information. Alternatively, it may add new functionality to let the users post comments about the places (e.g., hotels) recommended by the service. We observe that the data kept by the service may keep evolving, and the quality of the application code and the rules to infer the POIs may decay — these code or rules may require maintenances. Reviewing all the code or rules of the service can be time-consuming. On the other hand, a modification to such database may offer frequently. In this paper, we call a set of test cases as a test suite. An efficient approach is to apply regression test suites to assure the validity of the inferences. If there is any unexpected change witnessed by the regression test suite, a follow-up review or repair can be scheduled. In this way, it reduces the amount of artifacts to be reviewed at the expense of re-running regression test cases.

Observation 1: Data maintenance of a service may render the service to provide unintended behaviors to its service consumers, making the code maintenance of the service necessary. Even if the code of the service has not been modified, regression testing identifies software artifacts that may require software maintenance.

To assure such a frequent update of the location-based services does not adversely affect the quality of the service, test engineers may perform *high-frequency* regression testing to gain confidences on their services by detecting the presence of software anomalies.

Test case prioritization is an aspect of regression testing that permutes a test suite with the view of maximizing a test goal [21][29]. For the detection of software anomalies, it can be the rate of fault detection, i.e., how early the execution of the permuted test suite exposes the regression faults that are detectable by the unordered test suite [7][8][21][29]. In particular, test case prioritization does not discard any test cases, having the advantage of not impairing the fault detection ability of the test suite as a whole.

White-box test case prioritization techniques are not quite suitable for testing services with (soft) real-time constraint in general and location-based services in particular. Researchers have extensively studied code-coverage based techniques for white-box test case prioritization [7][8][14][27][29]. These techniques use a heuristics, namely faster code coverage on an application under test lead to a faster rate of fault detection, to reorder test cases.

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

† 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.

However, coverage-based techniques often require instrumentation or execution profiling to collect the code coverage information, which usually incur significant execution overheads. Nonetheless, a service-based application may use third-part services subject to certain timeliness constraints. Such an overhead may modify the Quality of Service (QoS) property of the application significantly. Even through a service (composition) under test can be tested, yet the actual service composition may use another set of QoS properties, and the representativeness of the test suite may not be useful. In other words, the results of the test may not be valid for such a service composition that is subject to end-to-end timeliness constraint.

Apart from using the white-box techniques, test engineers may use black-box information to guide the prioritization. One kind of black-box information is specification, but real-world software specifications can be inconsistent with the application under test [13] or unavailable, which makes such information hard to use. Furthermore, services are usually exposed with their public interfaces only.

On the other hand, the input and output information of the test cases can be collected without affecting the timing constraint of the service. For instance, Mei et al. [16] proposed techniques that reorder regression test cases based on the tags encoded in the input and output XML messages of a service under test over a test suite.

Many location-based services store POI information in advance for user enquiry. For instance, test engineers may consider that there is a POI database associated with the service under test, and apply testing techniques (e.g., [3]) for database-driven applications. They may focus their limited testing resources on the parts of spatial space represented by the POIs rather than the complete geospatial space.

Observation 2: Points of interest (POIs) of a service under test are correlated by their location information. Using such correlation information and applying Regularity Hypothesis about the behavior of the service help partition test cases in a regression test suite. Regression techniques may use such partition information to improve their efficiency and effectiveness.

A key observation is that the locations encoded in POIs are interrelated. That is, the contents of the table rows in the POI database or contents of the same tag in similar XML messages are correlated. For instance, the GPS locations of two hotels located on the same block of street are very close to one another.

Our insight is to use the distributions of POIs encoded in a regression test suite to permute test cases in the test suite. For instance, test engineers may choose to test the "POI clusters" from largest to smallest in a round robin manner [16]. In essence, it applies the regularity hypothesis to sample test cases from each partition (e.g., a POI cluster) for a service or a service composition. In this paper, we report the performance of our techniques [28] in terms of the average rate of fault detection (APFD) [7], which has not been reported. Unlike [28], this paper reports the effectiveness results using test suites of different scales, and analyzes the performance of our techniques across different sizes of test suites.

The empirical result on the case study shows that our techniques are on average more effective than random ordering, and POI-aware techniques are on average more effective than techniques that permute test cases based on input information. Moreover, we observe that the APFD performance of all techniques (including random ordering) improves as the size of a regression test suite increase.

The contribution of the paper is twofold: (1) We report a regression testing experiment on LBS and analyze the results in terms of APFD. To the best of our knowledge, it is the first experiment that reports this dimension. (2) We identify two important observations for service testing.

We organize the rest of paper as follows: we first analyze one example to review our POI-guide prioritization techniques in Section II. Then we present our family of POI-aware prioritization techniques in Section III. After that, we present our empirical study on a location-based recommender composite service system in Section IV. Finally, we describe the related works in Section V, which is followed by a conclusion in Section VI.

II. Motivating Study

In this section, we present an example using the data and code extracted from our *City Guide* location based service (case study).

The *City Guide* location based services recommends the best hotels (and restaurants, cinemas, etc) based on the location of the user as well as user preferences. The *City Guide* LBS service is a composite services that depends on a case-based reasoning [2] service and a data service. The case-based reasoning service chooses the best POI for recommendation based on case similarity. Each case contains the GPS location, user preferences and the identity of the best matched POI. The cases are stored in a MySQL database wrapped within the data service. We will next show why POI information can be effective to detect regression faults with an example.

```
public class GpsLocationSimilarity {
    public double compute(Gps loc1, Gps loc2){
        // Compute distance here...
        double similarity =
                distance / MAX_DISTANCE;
        //Bug, should be
        //       1 - distance / MAX_DISTANCE;
        return similarity;
    }
}
```

Figure 1. Faulty code with missing operation

Consider the fault shown in Figure 1. The program has a fault in the method computing the similarity between two GPS locations. The GPS location similarity measures how much two GPS locations are near to each other, which the value should decrease as two GPS locations are far away from each other. The GPS location distance is the earth surface distance between two GPS locations. The original version program uses the formula `similarity = 1 - distance / MAX_DISTANCE` to compute the similarity, but the faulty version directly uses the distance as the location similarity. As a result, the trend of similarity is reversed — it increases when two GPS locations get far away to one another.

Figure 2 shows two test cases. Each test case is composed of a sequence of GPS locations and one of these locations is marked as the start point. GPS locations are used to query the LBS to retrieve the nearby POIs. As the total number of POIs kept in a data service can be very large, it is unlikely for the LBS to consider all the POIs on a map. To return a reasonable list of POIs, those POIs nearer to the location being queried are more likely to be considered. Suppose that we consider merely the POIs located within one kilometer away (approximately 0.09 degree difference along the longitude or the latitude on the earth surface) from the GPS location being queried. Further suppose that we are interested in the nearest three POIs to each GPS location.

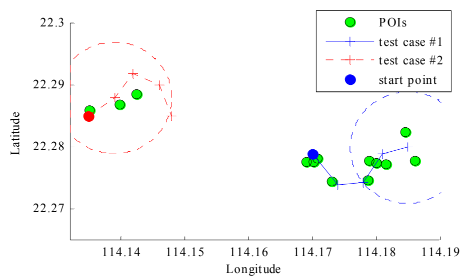

Figure 2. Test cases with different POI coverage

- **Test case #1.** When the location-based service is queried by using the last GPS location of the test case `(114.185,22.28)`, six nearby POIs within one kilometer will be considered by the LBS. The similarity between each POI and the location is computed. Owing to the fault, three farthest POIs rather than the three nearest POIs are returned. And thus, the fault manifests itself to become a failure.
- **Test case #2.** In contrast, for the locations in test case #2, within one kilometer, there are at most three POIs, and thus they are selected irrespective to their degrees of similarity with the user locations. Hence, the results of executing test case #2 are the same for both faulty and original pieces of code.

The example illustrates that the closer a GPS location sequence to the POIs, the more effective this sequence is to detect faults in the location-based service. To achieve better regression testing effectiveness, we proposed in [28] several techniques following this observation.

III. Review of Prioritization Techniques for LBS

This section reviews our proposed metrics [28] for test prioritization techniques for location-based services.

We have proposed five techniques using five different quantitative metrics in [28], namely sequence variance (*Var*), centroid distance (*CDist*), polyline entropy (*Entropy*), polyline distance (*PDist*) and POI coverage (*PCov*).

Our proposed techniques can be roughly classified in two groups in term of whether POI information is considered: input-guided techniques and POI-aware techniques. There are different concerns for the two groups. On the one hand, our previous work has demonstrated that the more diverse a sequence is, the more effective it can be to detect the faults [26]. Hence, input-guided techniques target to prioritize a test suite in descending order of the "diverseness" of test inputs. On the other hand, following the above-mentioned observation, POI-aware techniques aims at prioritizing test cases that are closer to POIs or cover more POIs.

To review our technique formally, we firstly formulate some concepts. Suppose $T = \{t_1, t_2, \ldots, t_n\}$ is a regression test suite with n test cases. Each test case t is sequence of GPS locations: $t = \langle l_1, l_2, \ldots, l_{|t|}\rangle$. Each GPS location is a pair of real number representing the longitude and the latitude: $l_{i,j} = (long_i, lat_i)$. An ordered test sequence S is defined as $S = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_n}\rangle$, where $\langle i_1, i_2, \ldots, i_n\rangle$ is a permutation of $\langle 1,2,\ldots,n\rangle$. POIs are a set of m GPS locations: $P = \{p_1, p_2, \ldots, p_m\}$. Each POI is also denoted by a pair $p_k = (long_k, lat_k)$.

A goal evaluation function g is a function that accepts a test goal G and test sequence S, and returns a real number R which indicates how well S scores with respect to G. Without loss of generality, we assume that a larger g value indicates a better satisfaction of G by S.

In our proposed techniques, the test sequence S is computed by sorting the test cases with respect to the quantitative metric M measured on each test case t_i in T. This can be described as a sorting function: $S = sort(T, M)$, and typically the sorting function is either in ascending order $sort_{asc}$ or descending order $sort_{desc}$. Moreover, our proposed techniques use different metrics guiding the sorting progress to obtain a desirable value of $g(G, sort(T, M))$, where M is either a POI-aware metric $M_p(t, P)$ or an input-guided metric $M_s(t)$.

The following five sub-sections discuss our proposed techniques in detail.

A. *Sequence Variance (Var)*

Sequence variance is an input-guided metric that evaluates the variance of a GPS location sequence. It is defined as the second-order central moment of the sequence:

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

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 may indicate a higher variety. The sorting function $sort_{desc}(T, Var(t))$ is used for this test case prioritization technique.

B. *Polyline Entropy (Entropy)*

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

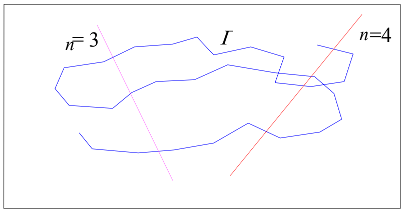

Figure 3. Illustration of the number of intersect points

The entropy of a curve comes from the thermodynamics of curves developed by Mendès France [19] and Dupain [6]. Consider a finite plane 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 for D to intersect Γ in n points, as illustrate in Figure 3. The entropy of the curve is given by [19] as

$$H[\Gamma] = -\sum_{n=1}^{\infty} P_n \log P_n \tag{2}$$

A classical computation for the maximal entropy gives [19]

$$P_n = e^{-\beta n}\left(e^{\beta} - 1\right) \tag{3}$$

with

$$\beta = \log\left(\frac{2L_\Gamma}{2L_\Gamma - C_\Gamma}\right) \tag{4}$$

Then, by combining (2), (3) and (4), we can obtain the function of the entropy of a plane curve Γ: [19][6]

$$H(\Gamma) = \log\left(\frac{2L_\Gamma}{C_\Gamma}\right) \tag{5}$$

Similarly, we define the polyline entropy of a test case as

$$Entropy(t) = \log\left(\frac{2L_t}{C_t}\right) \tag{6}$$

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 polyline of higher complexity. We sort the test cases in descending order of entropy, that is, we use the formula $sort_{desc}(T, Entropy(t))$.

C. *Centroid Distance (CDist)*

The *centroid distance* represents the distance from the centroid of a GPS location sequence to the centroid of the POIs. Since POI information is used in the computation, this metric is a POI-aware metric. It directly measures how far a test case is from the centroid of the POIs.

We formulate this metrics as follow:

$$CDist(t, P) = \|\bar{l} - \bar{p}\| \tag{7}$$

where $\bar{p} = (\sum p_i)/m$ is the centroid of all POIs. Moreover, a shorter distance between a sequence and the POIs is more preferable to a longer distance, so the sorting function used with *CDist* is $sort_{asc}(T, CDist(t, P))$.

D. *Polyline Distance (PDist)*

As mentioned in the last section, we regard each test case as a polyline whose vertices are GPS locations. *Polyline distance* measures the mean distance from all POIs to this polyline. Let the function $dist(p, t)$ denote the distance from a point $p = (long, lat)$ to a polyline $t = \langle l_1, l_2, \ldots, l_{|t|}\rangle$. Then, the polyline distance of a test case t is give by

$$PDist(t, P) = \frac{\sum_{i=1,2,\ldots,|P|} dist(p_i, t)}{|P|} \tag{8}$$

Similar to *CDist*, a shorter distance is more desirable for prioritization. We use $sort_{asc}(T, PDist(t, P))$ as sorting function for *PDist*.

E. *POI Coverage (PCov)*

POI coverage is a metric to evaluate the impact of POIs on each test case. Consider a test case t. To compute the *PCov* value of t, we first compute the distances from all POIs to the polyline represented by t as *PDist*. Then, we use a threshold value α to classify whether a POI is covered by the polyline. That is, if the distance from this POI to the polyline is no greater than α, we count this POI as "covered". Hence, this metric is given by the formula:

$$PCov(t, P) = \sum_{i=1,2,\ldots,|P|} f\left(dist(p_i, t)\right) \tag{9}$$

where

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

Following the motivating studies in Section II, when a test case covers more POIs than another test case, the former one can arguably reveal more faults potentially. Hence, we use $sort_{desc}(T, PCov(t, P))$ for this technique.

We summarize all the five techniques used in Table 1. We note that each technique randomly resolve tie cases.

TABLE 1. TECHNIQUES USED IN OUR STUDY

Name	Type	Metric Formula	Ordering
Random	Random selection	–	–
Var	Input-Guided	Equation (1)	Descending
Entropy		Equation (6)	Ascending
CDist	POI-Aware	Equation (7)	Descending
PDist		Equation (8)	Ascending
PCov		Equation (9)	Descending

IV. EMPIRICAL STUDY

In this section, we report an empirical study that evaluates the effectiveness of the techniques that are shown in Table 1.

A. *Research Questions*

RQ1: *Are the input guided black-box test case prioritization techniques more cost-effective than random ordering at all test suite sizes to detect software faults?*

RQ2: *Is the POI information helpful for guiding test case prioritization to detect software faults?*

B. *Subject Programs and Test Suites*

We use a location-based program *City Guide* in the experiment. This application consists of a number of web services. It includes a client service that collects the GPS data and displays the Google Map on the Android platform for mobile phone. At the server side, it has a service (the subject program of the experiment) that communicates with a case base engine, constructs a mashup script, and passes the constructed mashup script to Google Map to visualize the POIs. We note that both Google Map and the case base engine are third party services. The function of the Android client is merely to pass the GPS data to the server and display the Google Map. We exclude these three services as the subject programs in the study.

To generate faulty versions objectively, we used MuClipse and used all mutation operators described in [15]. Each generated mutant was regarded to be a faulty version of our subject program, and we deemed that the original version of the subject program as the golden version. Table 2 shows the descriptive statistics for the subject program.

TABLE 2. SUBJECT PROGRAM DESCRIPTION

Faulty Version	LOC	Test Pool Size
35	3289	2000

Although the mutant operators generated a large number of faulty mutants, not all of them wew suitable for testing purpose. We excluded all those mutants that cannot be revealed by any test case as well as those who were detected by more than 20% test cases [7]. At the end, 35 remaining mutants were selected for the experiment.

In terms of service testing, we note that the experiments in [17] used a suite of eight BPEL programs with 60 faults. We believe that the scale of our case study may not be smaller than the above-mentioned work.

C. *Experimental Environment*

We carried out the experiment on a Dell PowerEdge 1950 server serving a Solaris UNIX. The server was equipped with 2 Xeon 5355 (2.66Hz, 4 core) processors with 8GB physical memory.

D. *Effectiveness Metrics*

In this experiment, we measured how quickly a test suite can detect faults. We used APFD [8][17][28] as the metric. APFD measures the weighted average of the percentage of faults detected over the life of the suite. 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 for S is given by the following equation [8]:

$$APFD = 1 - \frac{TF_1+TF_2+\cdots+TF_m}{nm} + \frac{1}{2n} \quad (10)$$

E. *Experiment and Discussions*

Our case study *City Guide* contained a location-based services and a case-based reasoning engine. To set up the program for testing, we collected over 100 real hotels on the Hong Kong Island of Hong Kong as POIs and then initialized the case base engine with these POIs. Then, we randomly generated a test pool containing 2000 test cases which were reasonable GPS location sequences on the Hong Kong Island. All test suites used in the experiment were selected from this test pool.

Since the size of test suites intends to affect the effectiveness of testing, we used a wide range of test suite sizes to evaluate the performance of our proposed techniques. In particular, the sizes of test suite we used are 128, 256, 512 and 1024. For each size, we randomly extracted the test suite by 50 times and prioritized each of the 50 test suites using the six techniques described in Section III to obtain the averages. Hence, we computed a total of 200 APFD values for each technique and obtain 1200 prioritization results. We analyzed these results to answer the proposed research questions.

1) *Answering RQ1*

For each technique, we calculated the APFD results on all the faulty versions, performed analysis of variance (ANOVA), and drew four box-and-whisker plots for test suites of size 128, 256, 512 and 1024, as shown on the left hand side of Figure 4. For each box-and-whisker plot, the *x*-axis represents prioritization techniques and the *y*-axis represents their APFD distributions for all 50 test suites of a given size. 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 medians of the two groups differ at the 5% significance level. Having compared the distributions of different prioritization techniques, we further performed the multiple-comparison procedures to find those techniques whose means differ significantly from those of others at the 5% significance level, which are shown on the right hand side of Figure 4. The multiple comparison procedure shows the distribution of APFD values of each technique as a horizontal line with a dot in the center, which denotes the mean value. If the lines of two techniques do not overlap, then their mean value differ significantly at the 5% significance level.

From the subfigures (1), (3), (5), and (7) of Figure 4, we observe that, the two input-guided techniques perform better than random ordering. More specifically, the *Entropy* is consistently better than random ordering significantly for all sizes of test suite. The median value of *Var* technique is better than random ordering significantly for test suite size 128 or 256. For suite size 512 or 1024, the *Var* technique is slightly better than random ordering, but the difference is not statistically significant. If we compare their mean values

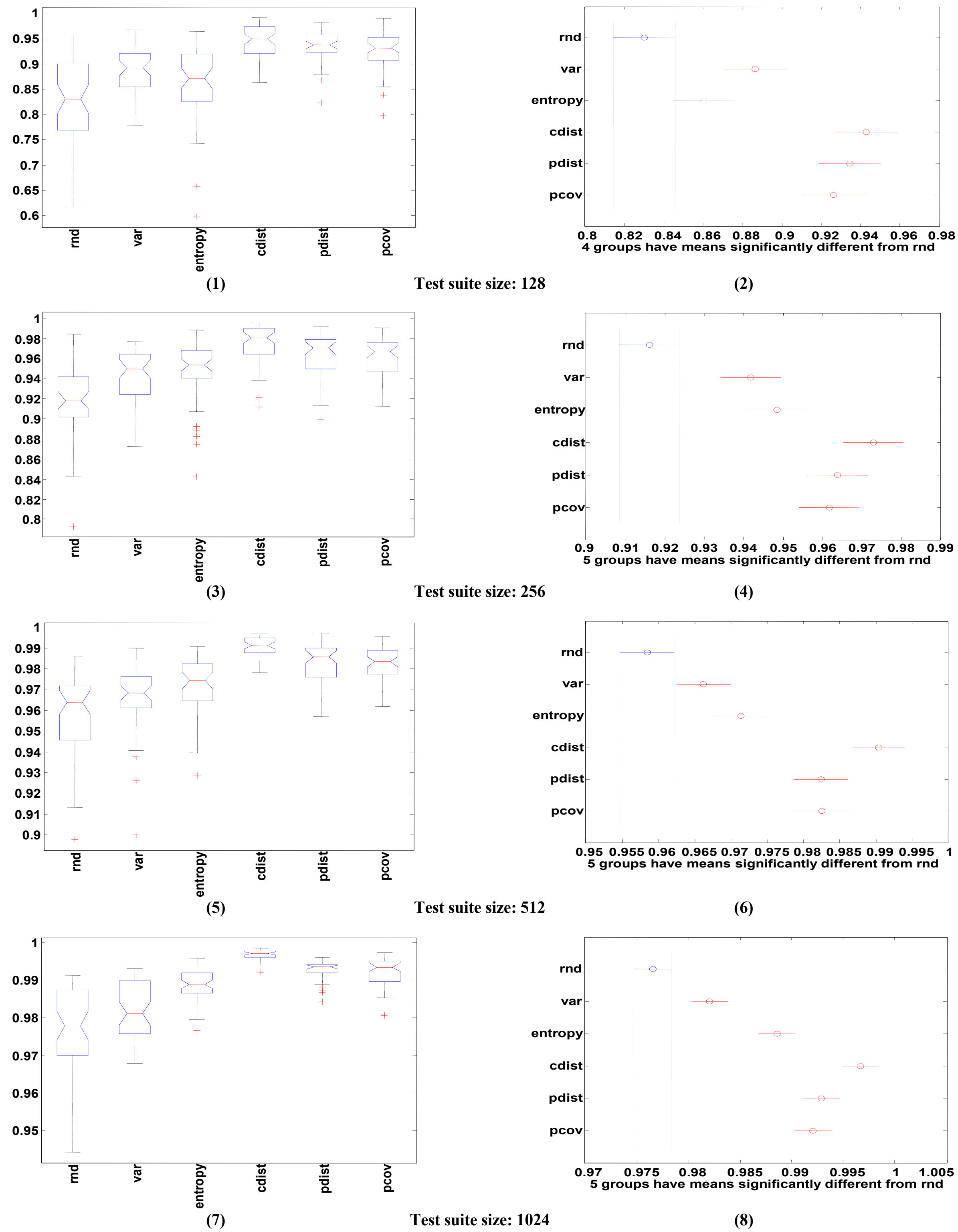

Figure 4. APFD Distribution and Multiple Comparisons for Different Size of Test Suites

from the subfigure (2), (4), (6) and (8) of Figure 4, we find that both input-guided techniques perform consistently better than random ordering significantly for all sizes of test. Thus, we can answer RQ1 that the input-guided test case prioritization is more effective than random ordering.

2) Answering RQ2

From the box-and-whisker plots in Figure 4, we observe that the three POI-aware techniques (*CDist*, *PDist* and *PCov*) are always significantly better than both the random ordering and the two input-guided techniques (*Var* and *Entropy*) at the 5% significance interval. Moreover, the results are consistent across different sizes of test suites.

We also examine the multiple comparison analysis results. There is no overlapping between POI-aware techniques and random ordering and between the POI-aware techniques and *Var* technique, the result shows that the POI technique are significantly better than random ordering and *Var* in terms of the mean APFD at the 5% significance level.

Moreover, this result is robust for all test suite sizes. The mean APFD of POI-aware techniques are better than *Entropy* statistically when the suite size is 512. For the other suite sizes, POI-aware techniques also have higher mean APFDs than *Entropy*, although the difference is less significant. We observe that the APFD value of each technique improves as the size of test suite increases.

Lastly, we find that median APFD values of the POI-aware techniques are larger than 0.90 in all cases. It indicates that they can be effective.

3) Conclusion

In conclusion, our empirical study shows that all our proposed black-box prioritization techniques are more effective than random ordering for different sizes of test suites in terms of APFD. Furthermore, the three prioritization techniques guided by POI information are better than the two input-guided prioritization techniques and the random ordering in terms of mean APFD values at the 5% significance level. POI information can be helpful for guiding test case prioritization for the regression testing of location-based services than the other reported techniques.

F. *Threats to Validity*

In this study, we used a location based information service composition to evaluate our techniques. The subject program depends on a case-based reasoning service, a data service, and an Android client. It is possible for the LBS (subject) to use other reasoning engines like rule-based, ontology-based, or simply geospatial database. Thus, one may evaluate our POI–aware prioritization techniques on more location based information services with different architectures to strengthen the external validity of the experiments. The subject program is implemented in the Java language. However, since our techniques require black-box information only, they are applicable to LBS implemented in other programming languages. However, the corresponding set of faults will be different, which may produce different APFD values even using the same test pool.

A threat to internal validity is the correctness of our tools. We used Java (Eclipse) and Matlab to implement our tools for instrumentation, test suite prioritization, and results calculation. To avoid errors, we have reviewed and tested our tools with a few test cases and manually verify the results.

We measured the rate of fault detection by APFD, which has been widely used in previous experiments [7][12]. Using other metrics may give different results.

V. RELATED WORK

In previous studies, many test case prioritization techniques were proposed, many of which were coverage-based. Wong et al. [27] proposed to combine test suite minimization and test case prioritization to select cases based on their cost per additional coverage. Srivastava et al. [23] 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. Walcott et al. [25] proposed a time-aware prioritization technique based on a genetic technique to permute test cases under the given time constraints. They all required the collection of white-box statistics.

Researchers also investigated the problem of regression testing of service-oriented applications. In [17], 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 conflicts faced by prioritization. In [17], 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. Adaptive random testing [4][5] aims at improving the fault detection rate of random test case generation by evenly spreading the test cases across the input domain. Jiang et al. [12] proposed a family of adaptive random test case prioritization techniques that tried to spread the test cases as evenly as possible across the code space to increase the rate of fault detection. Our POI-aware techniques use no code coverage.

Locations-based service is a popular application that benefits both the mobile network operators and the end users [20]. At one side, the operators may offer more value-added location based services to generate revenues from their infrastructure. On the other side, the end users can enjoy better user experience and high quality service with the help of LBS. There are many standards [1] and techniques for location-based services. In future, we plan to generalize our techniques so that they can be applicable to a broader spectrum of geospatial-based applications.

VI. CONCLUSION

Location-based information services intelligently provide personalized functions based on location information. They in general have timing constraints to be satisfied and some sub-services are not within the control of the test engineers of the service under test. In practice, such a service may modify existing features, add new features, or correct bugs. To assure the modifications not affecting the consumers of the service, test engineers may perform regression testing to detect faults.

We observe that maintaining data for a service may lead the service to provide unintended behaviors to its service consumers. It may lead to subsequent service maintenance. Regression testing can be an efficient approach to identifying such service problems even though the code has not been modified. Moreover, points of interest are correlated by their location information. They provide natural partition boundary for one to apply regularity hypothesis to support service testing.

In this paper, we have reported an empirical study to evaluate the effectiveness of our black box POI-aware test case prioritization techniques. The empirical result has shown that the POI-aware prioritization techniques are stable in terms of APFD as they perform consistently well with different sizes of test suites and are significantly better than random ordering or input-guided prioritization techniques statistically. We have also found that POI-aware prioritization techniques are effective in terms of APFD.

In future, we plan to expand our study to other context-aware services having more than one kind of contextual information.

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] A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications* 7: 1, 39−52, 1994.

[3] W. K. Chan, S. C. Cheung, and T. H. Tse. Fault-based testing of database application programs with conceptual data model. In *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*, pp. 187−196, 2005.

[4] 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 (JSS)*, 83 (1): 60-66, 2010.

[5] 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.

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

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

[8] 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.

[9] http://googlemobile.blogspot.com/2009/06/places-directory-app-for-android.html. Last accessed February 2010.

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

[11] 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.

[12] 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.

[13] B. Jiang, T. H.Tse, W. Grieskamp, N. Kicillof, Y. Cao and X. Li. Regression testing process improvement for specification evolution of real-world protocol software. To appear in *Proceedings of the The 10th International Conference on Quality Software (QSIC 2010)*. IEEE Computer Society Press, Los Alamitos, CA, 2010.

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

[15] 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.

[16] 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.

[17] 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.

[18] L. Mei, W.K. Chan, and T.H. Tse. Data flow testing of service choreography. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundation of Software Engineering (ESEC 2009/FSE-17),* pages 151–160, 2009.

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

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

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

[22] 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.

[23] 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*), 97–106, 2002.

[24] 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, Zurich, Switzerland, 2006.

[25] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time-aware 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.

[26] H. Wang and W. K. Chan. Weaving context sensitivity into test suite construction. In *Proceedings of the 24th International Conference on Automated Software Engineering (ASE 2009)*, pages 610–614, 2009.

[27] 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.

[28] K. Zhai, B. Jiang, W. K. Chan, and T. H. Tse. Taking advantage of service selection: a study on the testing of location-based web services through test case prioritization. In *Proceedings of the 8th IEEE International Conference on Web Services (ICWS 2010)*. IEEE Computer Society Press, Los Alamitos, CA, 2010.

[29] 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.