Towards Defining and Exploiting Similarities in Web Application Use Cases through User Session Analysis

Sreedevi Sampath
CIS
University of Delaware
Newark, DE 19716
sampath@cis.udel.edu

Amie L. Souter
Computer Science
Drexel University
Philadelphia, PA 19104
souter@cs.drexel.edu

Lori Pollock
CIS
University of Delaware
Newark, DE 19716
pollock@cis.udel.edu

Abstract

With the highly increased use of the web comes a significant demand to provide more reliable web applications. By learning more about the usage and dynamic behavior of these applications, we believe that some software development and maintenance tools can be designed with increased cost-effectiveness. In this paper, we describe our work in analyzing user session data. Particularly, the main contributions of this paper are the analysis of user session data with concept analysis, an experimental study of user session data analysis with two different types of web software, and an application of user session analysis to scalable test case generation for web applications. In addition to fruitful experimental results, the techniques and metrics themselves provide insight into future approaches to analyzing the dynamic behavior of web applications.

1. Introduction

Broadly defined, a web-based software system consists of a set of web pages and components that interact to form a system which executes using web server(s), network, HTTP, and a browser, and in which user input (navigation and data input) affects the state of the system. A web page can be either static, in which case the content is fixed, or dynamic, such that its contents may depend on user input. Dynamic analysis of web applications can provide information that is useful in many ways. For instance, monitoring of an application is used to provide information about the load of traffic of user requests on an application at different times of the day. Knowledge of the pages accessed by individual users is used to customize a web application for more personalization. Information about the dynamic behavior of the application under normal usage can be used for modeling the application for analysis, coupled with modeling based on static information. Logging of the dynamic behavior of a web application can be used for automatic test case generation [8, 21, 28].

In this paper, we describe our work in analyzing user session data. By making minimal configuration changes to a web server, data can be collected as a set of user sessions, each session being a sequence of URL and name-value pairs[1]. The collection of logged user sessions can be viewed as a set of use cases where a use case is a behaviorally related sequence of events performed by the user through a dialogue with the system [11]. By learning more about the usage and dynamic behavior of web applications through user session data analysis, we believe that some software development and maintenance tools can be designed with increased cost-effectiveness. Particularly, the main contributions of this paper are the analysis of user session data with concept analysis, discovering the commonality of URL subsequences of objects clustered in concepts, an experimental study of user session data analysis with two different types of web applications, and an application of user session analysis to scalable test case generation for web applications. In addition to fruitful experimental results, the techniques and metrics themselves provide insight into future approaches to analyzing the dynamic behavior of web applications through analysis of user session data.

2. Clustering via Concept Analysis

Concept analysis is a sound mathematical technique for clustering objects that have common discrete attributes[3]. Concept analysis takes as input a set O of objects, a set A of attributes, and a binary relation $R \subseteq O \times A$, called a *context*, which relates the objects to their attributes. To analyze user sessions using concept analysis we define the objects to represent user sessions, and the attributes of objects are represented by URLs. While a user session is considered to

[1]The name-value pairs are associated with GET/POST requests.

	GDef	GReg	GLog	PLog	GShop	GBooks	GMyInfo
us1	X	X	X				
us2	X	X	X		X		X
us3	X	X	X	X	X		
us4	X	X	X		X	X	
us5	X	X	X				
us6	X	X	X	X	X	X	

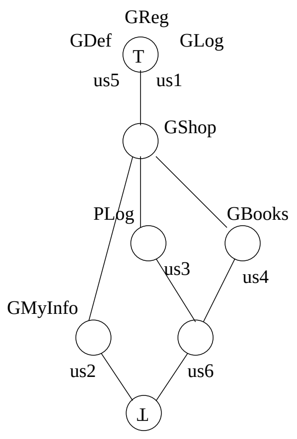


Figure 1. (a) Relation table and (b) concept lattices for test suite reduction

be a set of URLs and associated name-value pairs usually, we currently define a user session during concept analysis to be the set of URLs requested by the user, without the name-value pairs, and without any ordering on the URLs. This problem simplification considerably reduces the number of attributes to be analyzed, and results from our analysis of user sessions described in section 5 provide evidence to justify this simplification.

The relation table in Figure 1(a) shows the context for a set of user sessions for a portion of a bookstore web application [10] which we use for our experiments. Consider the row for the user, us3. The (true) marks in the relation table indicate that user us3 requested the URLs GDef, GReg, GLog, PLog and GShop. We distinguish a GET (G) request from a POST (P) request when building the lattice, since they are essentially different requests.

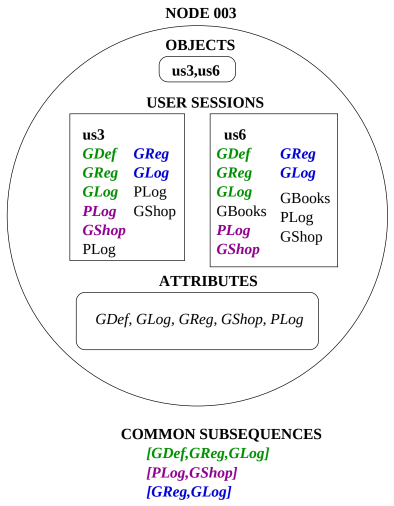
Concept analysis mutually intersects the user sessions for all observed use cases of the web application. The resulting intersections create a hierarchical clustering of the user sessions. Concept analysis identifies all of the concepts for a given tuple (O, A, R), where a *concept* is a tuple $t = (O_i, A_j)$ for which all and only objects in O_i share all and only the attributes in A_j The concepts form a partial order defined as $(O_1, A_1) \leq (O_2, A_2)$, *iff* $O_1 \subseteq O_2$. The set of all concepts of a context and the partial ordering form a complete lattice, called the *concept lattice*, which can be represented by a directed acyclic graph with a node for each concept and edges denoting the $\leq$ partial ordering. Based on the original relation table, concept analysis derives the lattice in Figure 1(b) as a sparse representation of the concepts.

A user session *s* requests all URLs at or above the concept uniquely labeled by *s* in the lattice. Similarly, a URL *u* is accessed by all user sessions at or below the concept uniquely labeled by *u*. The $\top$ of the lattice denotes the URLs that are requested by all the user sessions. The $\bot$ of the lattice denotes the user sessions that access all URLs in the context.

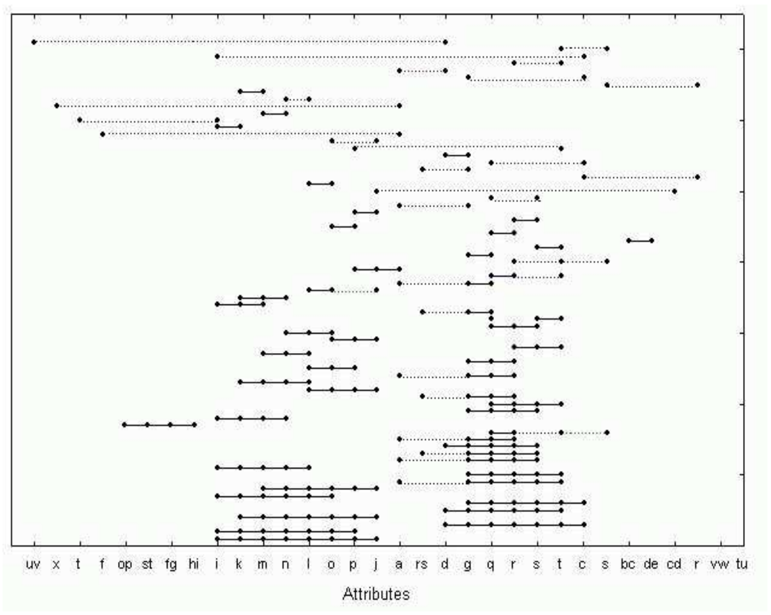
3. Examining Common Subsequences

Clustering of user sessions via concept analysis ensures that all objects in a node have the same set of common attributes. One question arises from clustering based on URL sets without considering the ordering of the URLs in each user session. Will user sessions represented by objects in the same concept node represent similar use cases? To answer this question, a measure of commonality of objects in terms of sequencing (i.e., ordering) of URLs is needed. We propose examining common subsequences as representative of partial use cases of the user sessions.

Figure 2(a) shows an example concept node, the attributes of that node, and the two user sessions represented by the objects of that node. The sequence of URLs for a user session are presented in columns, from left to right. The set of subsequences common to both of these user sessions are indicated below the node. For each concept node containing more than one object, we determine the longest common subsequence (LCS) of URLs among its objects. For different values of *k*, the set of unique subsequences of URLs of length *k* that are common to all the objects in the node is computed. This set is unique, in the sense, that occurrence of the subsequence *[PLog,GShop]* multiple times between the set of objects, is considered only once in the common subsequence set of size 2. Also, if a subsequence *[GDef,GReg,GLog]* is identified as common between objects of the node, then obviously all subsequences of *[GDef,GReg,GLog]* are also common. For the sake of fairness, we do not count subsequences of a larger sequence as smaller subsequences. However, for example, if the sequence *[GReg,GLog]* shows up in our results in addition to *[GDef,GReg,GLog]*, it is because *[GReg,GLog]* occurs as



(a)



(b)

Figure 2. (a) Example of common subsequences and (b) Spread of common subsequences over attributes for a node in Bookstore

a totally different subsequence from the larger subsequence in the use cases of the objects.

Another useful analysis is to examine the spread of the common subsequences of URLs of the objects of a concept node over the attribute space of that node. The graph in Figure 2(b) shows the spread of common subsequences over the attribute space of a node in the lattice for one of the applications we used. The x-axis shows the attributes of the concept node. For ease in showing URL ordering some of the attributes are repeated along the x-axis. This node has 37 attributes all of which are not shown on the axis (because they do not appear in any subsequence of size greater than 1). Continuous subsequences are represented by solid lines. A dotted line between two points, denotes that only the points form the subsequence. Only subsequences of size greater than one are shown in the graph. In this example graph, medium size subsequences cover some of the attributes and other attributes are covered by smaller subsequences. This spread of attribute coverage by common subsequences helps to provide some sense of the overall commonality of the use cases represented by different objects put into the same concept node based only on common sets of URLs.

In section 5.3, our experiments provide evidence that concept analysis with single URLs as attributes clusters objects together such that they have both the same set of attributes and large common partial use cases. In the next section, we describe how clustering these user sessions can be used in scalable test case generation.

4. Application to Test Case Generation

User session based testing exploits the ability of a web server to log user sessions for automatic test case generation. Our key insight to obtaining a scalable approach is to formulate user session based test case generation in terms of concept analysis. Existing incremental concept analysis techniques [9] can be exploited to analyze the user sessions on the fly, and continually minimize the number of maintained user sessions.

In our initial work, we developed a heuristic for selecting a subset of user sessions to be maintained as the current test suite, based on the current concept lattice. Given a context with a set of user sessions as objects O, we define the *similarity* of a set of user sessions $O_i \subseteq O$ as the number of attributes shared by all of the user sessions in O_i. Based on the partial ordering reflected in the concept lattice, user sessions labeling nodes closer to $\perp$ are more similar in their set of URL requests than nodes higher in the concept lattice.

Our heuristic for user session selection, which we call *test-all-exec-URLs*, seeks to identify the smallest set of user sessions that will still cover all of the URLs executed by the original test suite while representing the common URL subsequences of the different use cases represented by the original test suite. This heuristic is implemented as follows: The reduced test suite is set to contain a user session from each node next to $\perp$, that is one level up the lattice from $\perp$. We call these nodes *next-to-bottom* nodes. These nodes contain objects that are highly *similar* to each other. If the set of user sessions at $\perp$ is nonempty, those user sessions are also included. In our example in Figure 1, the original test suite is all the user sessions in the original context. The reduced test suite however contains only user sessions us2 and us6, which label the *next-to-bottom* nodes. By traversing the concept lattice to $\top$ along all paths from these nodes, we will find that the set of URLs accessed by these two user sessions are exactly the set of all URLs requested by the original test suite.

5. Experiments

In order to investigate the effectiveness and usefulness of user session clustering, and our heuristic for user session selection, we performed experiments utilizing a medium and large size application with real user sessions.

5.1. Research Questions

The experiments are designed to answer two questions with regard to user session clustering and selection for scalable test case generation: (1) How effective is the choice of using single URLs as attributes for clustering and is it reasonable to choose only one object from a concept node as the representative object? (2) How effective is the *test-all-exec-URLs* heuristic for selecting test cases for the current test suite? Our hypotheses with regard to these questions are:

1. The set of user sessions (i.e., objects) clustered into the same concept node will have a high commonality in the subsequences of URLs in their sessions. Thus, cost-effective clustering based on single URLs is reasonable, and only one representative from the next-to-bottom nodes can be chosen to be included in the current test suite.

2. In addition to covering all of the executed URLs of the original test suite, the user sessions (i.e., objects) of the next-to-bottom nodes (i.e., in the reduced test suite) execute a high percentage of the subsequences of URLs of the rest of the original test suite. We believe that this provides evidence that the original use cases are well represented by the reduced test suite.

5.2. General Methodology

We use an application from an open source e-commerce site [10] to experiment with applying concept analysis to user sessions to generate a reduced test suite. The application is a bookstore, where users can register, login, browse for books, search for specific books giving a keyword, rate the books, buy books by adding them to the shopping cart, modify personal information, and logout. The bookstore application has 9,748 lines of code, 385 methods and 11 classes. Since our interest was in user sessions, we considered only the code available to the user when computing these metrics, not the code that forms part of bookstore administration. The application uses JSP for its front-end and MySql database for the backend. The application was hosted on the Resin web server[22].

Emails were sent to various local newsgroups, and advertisements were posted in the university's classifieds webpage, asking for volunteers to browse the bookstore. We collected 123 user sessions, all of which were used in these experiments. Some of the URLs of bookstore mapped directly to the 11 classes/JSP files and the rest were requests for gif and jpeg images of the application. The size of the largest user session in bookstore was 863 URLs and on average a user session had 166 URLs.

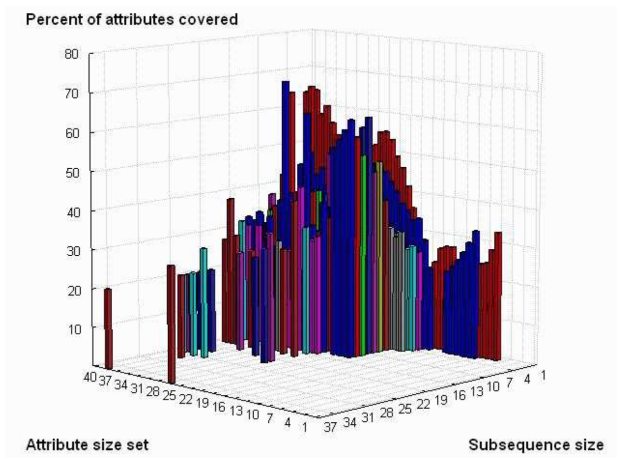
In addition to the bookstore, we also obtained user logs from a University of Delaware production application, uPortal[27], which is an abridged and customized version of the university's web presence, and has options for users to personalize the view of the campus web. The application is mainly open source and is written using Java, XML, JSP and J2EE. uPortal consists of 38,589 lines of code, 4233 methods, and 508 classes. The logs contained 2083 user sessions, which were also analyzed for the experimental study[2]. URLs collected for uPortal mapped directly to 6 of the JSP/Java files, but the data carried on them varied highly for each request. The size of the largest user session in uPortal was 407 URLs and on average a user session had 14 URLs.

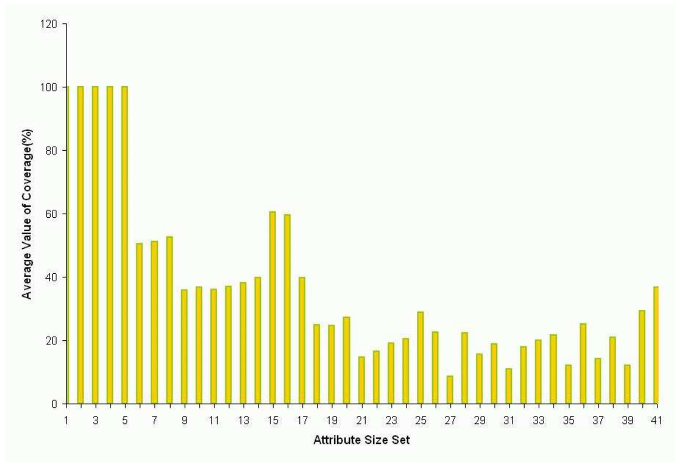
5.3. Commonality Among Attributes of a Concept

The focus of this experiment is to determine if objects clustered together in a concept node, in addition to having a set of common attributes, have a high commonality in the subsequences of URLs. If this is true, then only one object needs to be chosen from a given node for test case generation. Section 3 provided an introduction to the computation of common subsequences of a set of user sessions and our motivation to examine them. This section describes a new metric and experiments we performed towards quantifying the common subsequences of sets of user sessions.

Once common subsequences are generated as described in section 3, the nodes are grouped such that all nodes with the same number *n* of attributes are members of the attr-

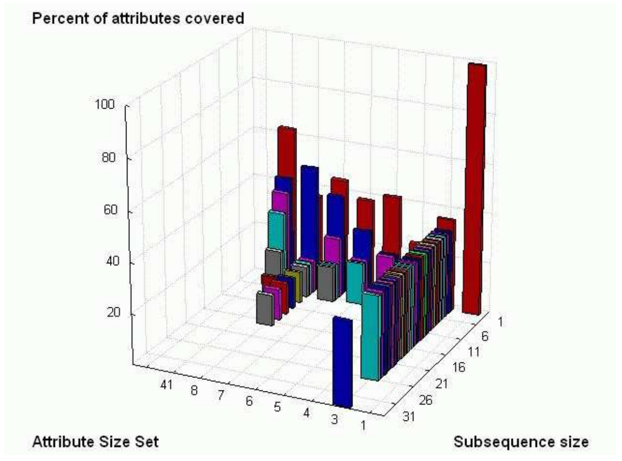
[2]We are currently creating a nonproduction uPortal version that maintains security of individual users' personal information for replay.

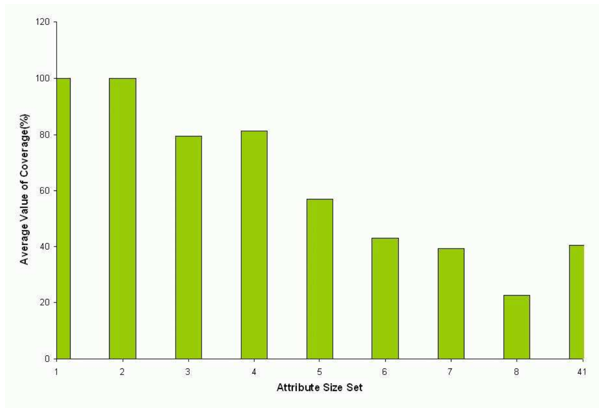




(a) **(b)**

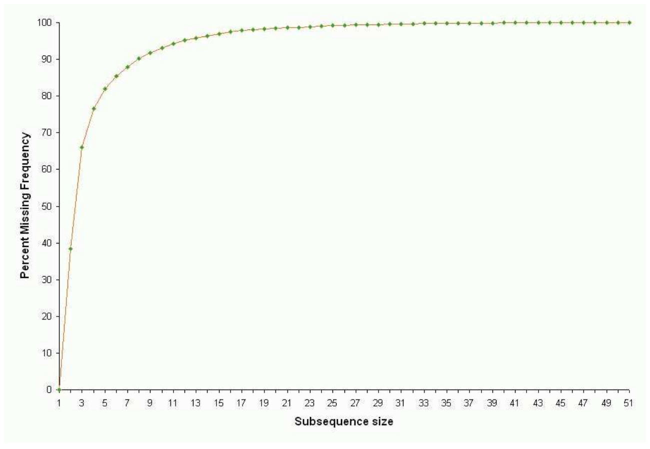
Figure 3. (a) Percent of attributes covered by different subsequence sizes and (b) Average percent of attributes covered by nodes in different attribute size sets for Bookstore



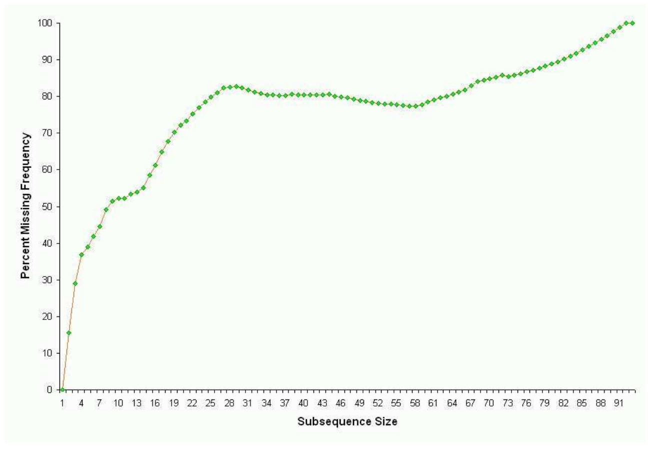


(a) **(b)**

Figure 4. (a) Percent of attributes covered by different subsequence sizes and (b) Average percent of attributes covered by nodes in different attribute size sets for uPortal



(a)



(b)

Figure 5. Percent subsequences not covered by *next-to-bottom* nodes for (a) Bookstore and (b) uPortal

size[n] set. This grouping gives a sense of the nodes' level in the concept lattice. Nodes with a low number of attributes will tend to be closer to $\top$.

We define the following metric: the percent of attributes (i.e., URLs) of a concept node that are included in (i.e., covered by) URL sequences of length *k* by objects (i.e., user sessions) in the node. For each attr-size set, the percent of attributes covered by each size subsequence is averaged over all the nodes in the set.

The results of computing this metric for the user sessions collected from bookstore and uPortal are shown in Figure 3(a) and Figure 4(a) respectively. The graphs show the percent of attributes covered by different subsequence sizes of nodes belonging to various attr-size sets. Coverage for subsequences of size 1 is not shown, because it is obviously 100%. As the attr-size increases, the percent coverage of attributes increases. These graphs are simply meant to demonstrate the trends of attribute coverage and attempt to illustrate that subsequences of varying sizes cover a reasonable percent of the attributes. For example, in Figure 3(a), size 10 subsequences across all attr-size sets cover between 27 to 62% of the attributes. In the bookstore, the largest subsequence, size 37, covers 21% of the attributes in one attr-size set and 30% in another. The computation produced similar results with the logs of uPortal (Figure 4(a)) – Size 6 subsequences across all attr-size sets covered between 13 to 33% of the attributes. The largest subsequence of size 33 covers 33% of the attributes.

To enable viewing the trend of percent attributes covered by nodes in different attr-size sets, the results were compiled in a different manner. First, the average percent of attributes covered (a_i) by each concept node *i*, over all subsequence sizes was computed. To be fair, the maximum longest common subsequence value for all nodes in a certain attr-size set is determined and is used in the above average, instead of averaging over the maximum size subsequence of each node. Then, an average percent coverage of the averages (a_i) for all the nodes in an attr-size set was computed. The results are shown in Figure 3(b) for bookstore and Figure 4(b) for uPortal. These graphs demonstrate that the average percent of attributes covered by nodes in various attribute size sets is quite high.

These results strengthen our first hypothesis that indeed there exists commonality in orderings of URLs between objects of a concept node and that these common subsequences cover a high range of percentage of the attributes of that node. Thus, clustering based on single URLs is reasonable for clustering similar use cases, and choosing one object from a given concept node as the representative test case will not result in loss of the attributes covered or the use cases represented by other objects in the node (Question 1).

5.4. Next-to-bottom Coverage of URL Orderings

The goal of this experiment is to support our hypothesis of choosing *next-to-bottom* nodes as the reduced test suite. We believe that such a selection will not cause large loss in representation of use cases associated with the remaining nodes in the lattice.

The *reduced set* is defined to contain the set of objects in the set of *next-to-bottom* nodes of the concept lattice. The difference between the objects that belong to the original test suite and the objects that belong to *reduced set* is called the *remaining set*. This experiment focused on determining the frequency of sequences of URLs that were present in the *remaining set* but missing in the *reduced set*. This metric is our measure to capture the 'loss of coverage' of use cases in the remaining set by the reduced set.

As can be observed for bookstore, in Figure 5(a), for subsequences of size 2, 38.37% of subsequences are missing, 66% of only size 3 subsequences are missing and 76.66% of just size 4 subsequences are missing. For uPortal user sessions, results shown in Figure 5 (b), 15.6% of size 2 subsequences are missing, 29.1% of only size 3 and 36.9% of only size 4 are missing. The percent missing subsequences increases with the size of the subsequence, because it is less likely for two user sessions to share exactly the same long sequence of URLs as many short similar sequences. For uPortal, we observed that there were only a few distinct URLs in the application, and the lengths of the user sessions were relatively small (average of 14 URLs).

It appears that the *reduced set* seems to be lacking the exact same long subsequence present in the *remaining set* but a large number of smaller size subsequences are present. Due to the clustering done by concept analysis, the *reduced set* is guaranteed to have more distinct URLs than the *remaining set*. So even if the exact same long subsequence is absent there are bound to be other sequences (that cover URLs missing in the *remaining set*) that are present in the *reduced set* but absent in the *remaining set*. The absence of a relatively small number of subsequences in the *reduced set* and the assurance due to concept analysis that, a larger number of URLs are present in this set, makes it suitable to be considered for the reduced test suite, and thus moderately supports our second hypothesis (Question 2).

To summarize, the experiments performed in this section support our hypotheses for user session clustering and selection and our heuristic for test case generation. We have performed some preliminary coverage and fault detection studies of test suites created by these techniques and found very promising results. More complete results will be described in a future paper. We believe that the metrics defined and the techniques applied can also be used for other dynamic analysis of web applications.

6. Related Work

Concept Analysis and Clustering in Software Engineering. Snelting first introduced the idea of concept analysis for use in software engineering tasks, specifically for configuration analysis [13]. Concept analysis has also been applied to evaluating class hierarchies [25], debugging temporal specifications [1], redocumentation [14], and recovering components [7, 15, 26, 24]. Ball introduced the use of concept analysis on test coverage data to compute dynamic analogs to static control flow relationships [2]. The binary relation consisted of tests (objects) and program entities (attributes) that a test may cover.

Similar to concept analysis is cluster analysis in which many techniques exist [12]. Such techniques are based on finding groups of clusters in a population of objects, where each object is characterized by a set of attributes. Cluster analysis algorithms use a dissimilarity metric to partition the set of objects into clusters.

To improve the accuracy of software reliability estimation [20], cluster analysis has also been utilized to partition a set of program executions into clusters based on the similarity or dissimilarity of their profiles. It has been experimentally shown that failures often correspond to unusual profiles that are revealed by cluster analysis. Dickinson et al. have utilized different cluster analysis techniques along with a failure pursuit sampling technique to select profiles to reveal failures. They have experimentally shown that such techniques are effective [5, 6]. Clustering has also been used to reverse engineer systems [4, 18, 19, 29].

Test Case Generation. Several tools exists that provide automated testing for web applications such as WebKing [28] and Rational Robot [21]. These tools function by collecting data from users through minimal configuration changes to a web server. The data collected can be viewed as user sessions, which is a a collection of user requests in the form of URL and name-value pairs. To transform a user session into a test case, each logged request of the user session is changed into an HTTP request that can be sent to a web server. A test case consists of a set of HTTP requests that is associated with a particular user session. Different strategies are applied to construct test cases for the collected user sessions. In these tools, test case generators are based on selecting most popular paths in web server logs. Studies have shown promising results that demonstrate the fault detection capabilities and cost-effectiveness of user session-based testing [8]. They showed that the effectiveness of user session techniques improves as the number of collected sessions increases. However, the cost of collecting, analyzing, and storing data will also increase.

Recently, analysis tools have been developed that model the underlying structure and semantics of web-based programs. With the goal of providing automated data flow testing, Liu, Kung, Hsia, and Hsu [16] developed the object-oriented web test model (WATM). They utilized this model to generate test cases, which are based on data flow between objects in the model. Their technique generates def-use chains as test cases, which require additional analysis in order to generate test cases that can be utilized as actual input to the application. They do not indicate how this step would be accomplished.

Ricca and Tonella [23] developed a high level UML-based representation of a web application and described how to perform page, hyperlink, def-use, all-uses, and all-paths testing based on the data dependences computed using the model. Their ReWeb tool loads and analyzes the pages of the application and builds a UML model. The TestWeb tool generates and executes test cases. However, significant intervention is required by the user for generating input.

Lucca et al. [17] recently developed a web application model and set of tools for the evaluation and automation of testing web applications. They developed functional testing techniques based on decision tables, which help in generating effective test cases. However, the process of generating test input in this manner is not automated.

7. Summary and Future Work

This paper has demonstrated that interesting usage patterns of a web application can be uncovered through concept analysis combined with common subsequence analysis. This is just a first step towards better understanding the dynamic behavior of web applications. We have shown how this kind of analysis can be used for scalable, automatic test case generation for this application domain. Our future work includes modifying the heuristic *test-all-exec-URLs* to consider degree of similarity between user sessions, exploring additional user session analyses that might be useful for software engineering tasks, and combining user session analyses with dynamic analysis of the actual program code, towards accurate static modeling of web applications. These combined efforts would provide a basis for creating software development, testing, and maintenance tools for reliable web applications.

Acknowledgements
We thank the U. of Delaware IT group for their aid in logging uPortal sessions.

References

[1] G. Ammons, D. Mandelin, and R. Bodik. Debugging temporal specifications with concept analysis. In *ACM SIGPLAN Conf on Prog Lang Design and Implem*, 2003.

[2] T. Ball. The concept of dynamic analysis. In *ESEC / SIGSOFT FSE*, pages 216–234, 1999.

[3] G. Birkhoff. *Lattice Theory*, volume 5. American Mathematical Soc. Colloquium Publications, 1940.

[4] D. Bojic and D. Velasevic. Reverse engineering of use case realizations in uml. In *Proceedings of the 2000 ACM symposium on Applied computing*, pages 741–747, 2000.

[5] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *Proceedings of the 23rd international conference on Software engineering*, pages 339–348. IEEE Computer Society, 2001.

[6] W. Dickinson, D. Leon, and A. Podgurski. Pursuing failure: the distribution of program failures in a profile space. In *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 246–255. ACM Press, 2001.

[7] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Trans on Soft Eng*, 29(3):210–224, Mar 2003.

[8] S. Elbaum, S. Karre, and G. Rothermel. Improving web application testing with user session data. In *Int Conf on Soft Eng*, 2003.

[9] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995.

[10] Open source web applications with source code. <http://www.gotocode.com>, 2003.

[11] I. Jacobson. The use-case construct in object-oriented software engineering. In J. M. Carroll, editor, *Scenario-based Design: Envisioning Work and Technology in System Development*, 1995.

[12] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[13] M. Krone and G. Snelting. On the inference of configuration structures from source code. In *Int Conf on Soft Eng*, 1994.

[14] T. Kuipers and L. Moonen. Types and concept analysis for legacy systems. In *Int Workshop on Prog Compr*, 2000.

[15] C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In *Int Conf on Soft Eng*, 1997.

[16] C.-H. Liu, D. C. Kung, and P. Hsia. Object-based data flow testing of web applications. In *Proceedings of the First Asia-Pacific Conference on Quality Software*, 2000.

[17] G. D. Lucca, A. Fasolino, F. Faralli, and U. D. Carlini. Testing web applications. In *International Conference on Software Maintenance*, 2002.

[18] C.-H. Lung. Software architecture recovery and restructuring through clustering techniques. In *Proceedings of the third international workshop on Software architecture*, pages 101–104, 1998.

[19] B. S. Mitchell, S. Mancoridis, and M. Traverso. Search based reverse engineering. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 431–438, 2002.

[20] A. Podgurski, W. Masri, Y. McCleese, F. G. Wolff, and C. Yang. Estimation of software reliability by stratified sampling. *ACM Trans. Softw. Eng. Methodol.*, 8(3):263–283, 1999.

[21] Rational Robot. <http://www-306.ibm.com/software/awdtools/tester/robot/>, 2003.

[22] Caucho resin. http://www.caucho.com/resin/, 2002.

[23] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proceedings of the International Conference on Software Engineering*, May 2001.

[24] M. Siff and T. Reps. Identifying modules via concept analysis. In *International Conf on Software Maintenance*, 1997.

[25] G. Snelting and F. Tip. Reengineering class hierarchies using concept analysis. In *SIGSOFT FSE*, 1998.

[26] P. Tonella. Concept analysis for module restructuring. *IEEE Trans on Soft Eng*, 27(4):351–363, Apr 2001.

[27] Uportal. <http://www.uportal.org>, 2004.

[28] WebKing. <http://www.parsoft.com>, 2004.

[29] T. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *Fourth Working Conference on Reverse Engineering (WCRE '97)*, 1997.