Introducing a Test Suite Similarity Metric for Event Sequence-based Test Cases

Penelope A. Brooks and Atif M Memon
Dept. of Computer Science
University of Maryland
College Park, MD, 20742 USA
{penelope,atif}@cs.umd.edu

Abstract

Most of today's event driven software (EDS) systems are tested using test cases that are carefully constructed as sequences of events; they test the execution of an event in the context of its preceding events. Because sizes of these test suites can be extremely large, researchers have developed techniques, such as reduction and minimization, to obtain test suites that are "similar" to the original test suite, but smaller. Existing similarity metrics mostly use code coverage; they do not consider the contextual relationships between events. Consequently, reduction based on such metrics may eliminate desirable test cases. In this paper, we present a new parameterized metric, $CONTeSSi(n)$ which uses the context of n preceding events in test cases to develop a new context-aware notion of test suite similarity for EDS. This metric is defined and evaluated by comparing four test suites for each of four open source applications. Our results show that $CONTeSSi(n)$ is a better indicator of the similarity of EDS test suites than existing metrics.

1. Introduction

Research in software testing has yielded a large number of automated model-based test case generation techniques [4, 9, 15, 18, 21]. Each of these techniques has the ability to generate test suites containing hundreds of thousands of test cases, which require significant resources to run, and for regression testing, rerun [14]. For this reason, research in test case selection and reduction has been growing in an attempt to shrink these test suites to a manageable size, while maintaining the "goodness" of the original suite.

Reduction techniques attempt to yield a test suite that is "similar" to the original suite in some ways, where similarity is usually determined by using metrics based on code (*e.g.*, obtained from branch, statement, and method coverage reports) executed by the original suite [6,7,12,16,17] or the set of faults detected. These metrics work well when applied to conventional software; they are not appropriate for event driven software (EDS) systems, such as software that uses a Graphical User Interface (GUI) front-end [14] because of the nature of EDS. EDS systems take sequences of events as input and change state; the execution of an event may be effected by the sequence of preceeding events handled thus far by the EDS. Consequently, test cases for EDS are composed of sequences of events designed specifically to test events in the context of preceding events. Existing test suite reduction metrics ignore context completely. Consequently, test reduction may eliminate a desirable (albeit redundant according to existing metrics) test case that tests an event in a specific context.

In this paper, we define a new parameterized metric, $CONTeSSi(n)$ (CONtext Test Suite Similarity) that explicitly considers the context of n preceding events in test cases to develop a new "context-aware" notion of test suite similarity. This metric is an extension of the cosine similarity metric used in Natural Language Processing and Information Retrieval for comparing an item to a body of knowledge, *e.g.*, finding a query string in a collection of web pages or determining the likelihood of finding a sentence in a text corpus (collection of documents) [2, 19, 20]. We evaluate $CONTeSSi(n)$ by comparing four test suites, including suites reduced using conventional criteria, for four open source applications. Our results show that $CONTeSSi(n)$ is a better indicator of the similarity of test suites than existing metrics.

The contributions of this work include:

- the first context based similarity metric for sequence-based test cases,
- application of the metric to reduced test suites, and
- an empirical study demonstrating the effectiveness of the metric.

The remainder of this paper is organized as follows. Section 2 describes the development and theory of the metric.

Section 3 contains a description of the empirical study performed to validate this metric, and its results are described in Section 4. Section 5 expands on work in related areas, such as determining similarity in information retrieval. Finally, conclusions and future work are presented in Section 6.

2. Computing test suite similarity

We now present the $CONTeSSi$ metric, its computation, and application. As a running example, we will show several test suites which can be used to test the Radio Button Demo GUI, shown in Figure 1. This GUI is often used to teach programming students how to develop a GUI containing radio buttons.

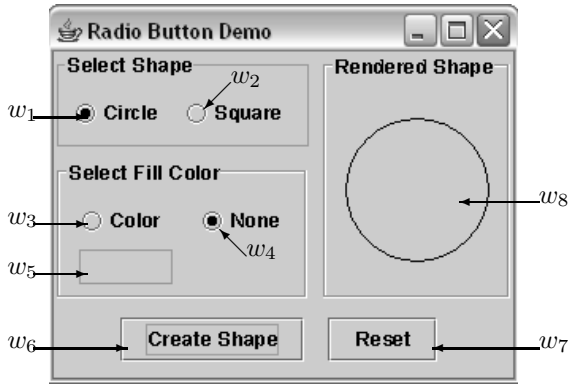

Figure 1. A simple GUI

A GUI is modeled as a set of *widgets* $W = \{w_1, w_2, ..., w_l\}$ (*e.g.*, buttons, panels, text fields) that constitute the GUI, a set of *properties* $P = \{p_1, p_2..., p_m\}$ (*e.g.*, background color and shape) for each of these widgets, and a set of *values* $V = \{v_1, v_2..., v_n\}$ (*e.g.*, red and square) associated with the properties. The state of a GUI can be specified at any time during its execution as a set S of triples (w_i, p_j, v_k), where $w_i \in W$, $p_j \in P$, and $v_k \in V$. Each GUI will contain certain types of widgets with associated properties.

The widgets for the Radio Button GUI (Figure 1), labeled w_1 through w_7, are those through which users can access the corresponding events (e_1 through e_7). In the *start state* of the GUI, `Circle` and `None` are selected; the text-box corresponding to w_5 is empty; and the `Rendered Shape` area (widget w_8) is empty. Event e_6 creates a shape in the `Rendered Shape` area according to current settings of $w_1 \ldots w_5$; event e_7 resets the entire software to its start state. The other events behave as follows:

- e_1 sets the shape to a circle; if there is a square in the `Rendered Shape` area, it is immediately changed to a circle,

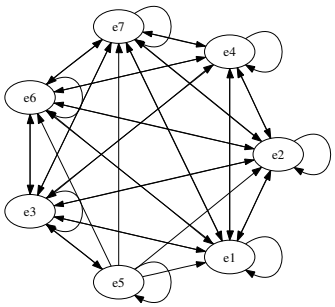

Figure 2. EFG for the Radio Button GUI

- e_2 is similar to e_1, except that it displays a square in the `Rendered Shape` area,
- e_3 enables the text-box w_5, allowing the user to enter a custom fill color, which is immediately reflected in the shape being displayed (if there is a shape there), and
- e_4 reverts back to the default color.

The class of GUIs that are considered in this research take input from a single user, have a fixed number of events and are deterministic; these GUIs can be represented by an *Event-Flow Graph* (EFG). Standard graph walking techniques can be used to reason about the model and generate test cases from the EFG [15]. An EFG is a specific model of the GUI for a particular application, representing all possible sequences of events that a user can execute on that GUI. Figure 2 shows the EFG for the GUI of Figure 1. Nodes in the EFG represent events, and directed edges represent the *event-flow* relationship between two events. That is, an edge in the graph from event e_1 to e_2 indicates that event e_2 may be invoked *immediately after* event e_1. EFGs are potentially cyclic, since events can typically be executed more than once during a session with an application.

Column 1 of Table 1 shows a test suite generated from this static model of the GUI. The test suites in the remaining columns were obtained using reduction techniques, and hence, are "similar" to the original suite. The popular HGS algorithm [6] was used to reduce the suite. Column 2 shows a suite reduced based on event pair coverage, which retains test cases that cover all unique pairs of events. The suite in Column 3 is reduced based on event coverage, which retains test cases that cover all unique events. The suites in columns 4-6 were reduced based on statement, method, and branch coverage, respectively. We note that event-pair coverage is the only reduction method that considers the context of a preceding event; however, it considers only a single event. Column 7 shows a suite consisting of test cases in which each test case executes one unique event. This suite will be used to illustrate the metric.

Original Tests	Event Pair Coverage	Event Coverage	Statement Coverage	Method Coverage	Branch Coverage	Illustrative Tests
e_1, e_6	e_1, e_4	e_1, e_4	e_7, e_1, e_6	e_1, e_3, e_5, e_6	e_1, e_4	e_1
e_2, e_6	e_1, e_3, e_5, e_6	e_7, e_2, e_3, e_5, e_6	e_7, e_2, e_6	e_7, e_2, e_3	e_7, e_1, e_6	e_2
e_7, e_1	e_7, e_2, e_3, e_5, e_6		e_2, e_3, e_5, e_6		e_7, e_2, e_6	e_3
e_1, e_4	e_3, e_5, e_4, e_3, e_6		e_3, e_5, e_4, e_3, e_6		e_2, e_3, e_5, e_6	e_4
e_2, e_3, e_5	e_2, e_4, e_7, e_2, e_6		e_7, e_2, e_3, e_5, e_6			e_5
e_7, e_1, e_6	e_7, e_1, e_6					e_6
e_7, e_2, e_6						e_7
e_2, e_4 e_7, e_2, e_6						
e_7, e_2, e_3						
e_1, e_3, e_5, e_6						
e_2, e_3, e_5, e_6						
e_3, e_5, e_4, e_3, e_6						
e_7, e_2, e_3, e_5, e_6						

Table 1. Example test cases yielded from several reduction techniques

Although these suites are similar to the original suite in terms of their respective reduction/similarity criteria, they are quite different when considering context. For example, the subsequence $< e_7, e_2 >$ appears four times in the original suite; $< e_2, e_3 >$ appears four times; $< e_1, e_6 >$ appears twice; each of these event subsequences appear in multiple contexts. Reduction does not consider preserving the importance of these frequencies and/or contexts.

Let us consider some notions of the similarity of two given suites. We can measure similarity based on the occurrence of events in both suites. If both suites contain exactly the same events, we could consider them to be very similar. However, that would allow us to say that a suite with 10 test cases of 5 events each, for a total of 50 distinct events, would be the same as a test suite with 1 test case with 50 of the same events. A better method of measuring similarity, however, would be to consider the frequency of events in the test suite. For example, counting the occurrence of e_3 in each test suite and using this count to compare the suites will apply a weight to the event and provides more information on the suite. We may use a vector to represent the count of each event in the suite, with each position in the vector representing the count of a single event. For our seven events e_1 to e_7 in the running example, this vector is produced: $< 5, 8, 7, 3, 5, 9, 6 >$, also shown in tabular form for all suites in Table 2(a). This is the basis of $CONTeSSi$.

Because EDS systems are highly reliant on the context in which events are executed, $CONTeSSi$ should return a value representing higher similarity when the same events occur in the same frequency and the same context between two test suites. As a starting point, consider the context for a single preceding event; a vector can be created based on the frequencies of event pairs observed in the test suite, rather than on a single event. In considering this context, the event pair coverage suite in Table 1 is expected to be more similar to the original suite than the event coverage suite, since the event pair coverage suite is created based on the existence of event pairs. Table 2(b) shows the count of each event pair for each suite. This is the basis of $CONTeSSi(n)$, for $n = 1$, since we are looking at events in the context of one other (previous) event.

Now if we extend this example to compute $CONTeSSi(2)$, we obtain the frequencies shown in Table 2(c). In general, as n increases, the frequencies for the event sequences decrease, as they appear less frequently in the test suites. Intuitively, comparing test suites on longer sequences will make it harder for the test suites to be similar. Therefore, if two test suites have a high similarity score with a larger n, they are even more similar than two suites being compared with a small n. By treating each row in Table 2(a), (b), or (c) as a vector, $CONTeSSi$ is computed as follows:

$$CONTeSSi(A, B) = \frac{(A \cdot B)}{(|A| \times |B|)} \qquad (1)$$

where A and B are the vectors corresponding to the two test suites, $A \cdot B$ is the dot product of the two vectors, *i.e.*, $\sum_{i=1}^{j}(A_i \times B_i)$ where j is the number of terms in the vector; and $|A| = \sqrt{\sum_{i=1}^{j}(A_i)^2}$. The value of $CONTeSSi$ lies between 0 and 1, where a value closer to 1 indicates more similarity. Hence, $CONTeSSi(n)$ is computed as shown in Equation 1, creating a vector for each suite, representing the frequencies of all possible groups of $n + 1$ events. The inclusion of n previous events will increase the number of terms in the vector, thereby increasing j. The values in Table 3 show the values of $CONTeSSi(n)$ for all our test suites, for $n = 0, 1, 2, 3$. From these values, we observe that if we ignore context, *i.e.*, use $n = 0$, most of the reduced suites are quite similar to the original, as indicated by the high (> 0.9) value of $CONTeSSi(0)$. However, the similarity between the test suites decreases as more context

Suite	e_1	e_2	e_3	e_4	e_5	e_6	e_7
Original	5	8	7	3	5	9	6
Evnt Pair	3	3	4	3	3	5	3
Evnt Cov	1	1	1	1	1	1	1
Stmt Cov	1	3	4	1	3	5	3
Meth Cov	1	1	2	0	1	1	1
Brnch Cov	2	2	1	1	1	3	2
Illus. Suite	1	1	1	1	1	1	1

(a) Frequency of individual events

Suite	e_1,e_3	e_1,e_4	e_1,e_6	e_2,e_3	e_2,e_4	e_2,e_6	e_3,e_5	e_3,e_6	e_4,e_3	e_4,e_7	e_5,e_4	e_5,e_6	e_7,e_1	e_7,e_2
Original	1	1	2	4	1	3	5	1	1	1	1	3	2	4
Evnt Pair	1	1	1	1	1	1	3	1	1	1	1	2	1	2
Evnt Cov	0	1	0	1	0	0	1	0	0	0	0	1	0	1
Stmt Cov	0	0	1	2	0	1	3	1	1	0	1	2	1	2
Meth Cov	1	0	0	1	0	0	1	0	0	0	0	1	0	1
Brnch Cov	0	1	1	1	0	1	1	0	0	0	0	1	1	1
Illus. Suite	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) Frequency of all event pairs

Suite	e_1,e_3,e_5	e_2,e_3,e_5	e_2,e_4,e_7	e_3,e_5,e_6	e_3,e_5,e_4	e_4,e_3,e_6	e_4,e_7,e_2	e_5,e_4,e_3	e_7,e_1,e_6	e_7,e_2,e_3	e_7,e_2,e_6
Original	1	3	1	3	1	1	1	1	1	2	2
Evnt Pair	1	1	1	2	1	1	1	1	1	1	1
Evnt Cov	0	1	0	1	0	0	0	0	0	1	0
Stmt Cov	0	2	0	2	1	1	0	1	1	1	1
Meth Cov	1	0	0	1	0	0	0	0	0	1	0
Brnch Cov	0	1	0	1	0	0	0	0	1	0	1
Illus. Suite	0	0	0	0	0	0	0	0	0	0	0

(c) Frequency of all event triples

Table 2. Frequency of n events in original and reduced test suites for Radio Button GUI example

(larger values of n) is considered for the events. The event- and method-coverage suites show relatively lower values of $CONTeSSi(3)$ because they retain very little context with only two test cases. The event-pair reduced suites have the highest value of $CONTeSSi(3)$, followed by statement and branch coverage reduced suites. Finally, the illustrative suite is very similar to the original when context is not considered ($CONTeSSi(0) = 0.956$). With the addition of context, however, the $CONTeSSi$ value is 0 due to the single event test cases. Even in this simple example, we see the value of the similarity metric.

To improve the context for events appearing at the beginning and end of test cases, we include two special sets of "events" called $INIT_n$ and $FINAL_n$. Without loss of generality, we add these events to all test cases. When computing $CONTeSSi(n)$, we prepend n $INIT$ events and append n $FINAL$ events to each test case. For example, in looking at event triples ($n = 2$), we will add two $INIT$ and two $FINAL$ events to each test case: $< INIT_0, INIT_1, e_2, e_3, e_5, FINAL_0, FINAL_1 >$ to glean that e_2 is at the start of $< e_2, e_3, e_5 >$, by obtaining the triple $< INIT_0, INIT_1, e_2 >$ and that e_5 is at the end of the sequence by obtaining $< e_5, FINAL_0, FINAL_1 >$. Due to space constraints, these events are not shown in Table 2.

	$Suite$					
n	Evnt Pair	Evnt Cov	Stmt Cov	Meth Cov	Brnch Cov	Illus. Suite
0	0.977	0.956	0.970	0.921	0.960	0.956
1	0.969	0.869	0.967	0.859	0.946	0
2	0.963	0.813	0.960	0.794	0.931	0
3	0.959	0.774	0.952	0.754	0.923	0

Table 3. CONTeSSi(n) value for $Suite$ compared to $Original$ for all Radio Button GUI example suites

3. Empirical study

To evaluate the quality of $CONTeSSi(n)$, we conducted an empirical study comparing several test suites on the basis of exisiting similarity metrics, such as statement coverage, method coverage, and event pair coverage. We

also used $CONTeSSi(n)$ to compare the same suites.

The goal of this study is to *evaluate a metric that measures the similarity of two test suites and to determine the quality of this metric*.

Restating this goal using the Goal Question Metric (GQM) Paradigm [3], the goal for this research is restated as follows:

> Analyze the **test suites**
> for the purpose of **comparison**
> with respect to **other test suites**
> from the point of view of the **tester/researcher**
> in the context of **event driven systems**.

From this goal, the following research questions are addressed:

1. **RQ1:** Does $CONTeSSi(n)$ agree with existing metrics in determining the similarity between suites, specifically relating to fault detection effectiveness?
2. **RQ2:** Does $CONTeSSi(n)$'s value improve for larger values of n?

Each of these research questions will evaluate the similarity metric by comparing existing test suites on coverage criteria and the $CONTeSSi$ metric. In most research and in practice, test suites are evaluated based on code coverage, fault detection, or both; the results of this study provide an objective method of comparing test suites without the need to run them.

The first question is focused on comparing the results of $CONTeSSi$ to the coverage of the suite, and further examining the relationship between the metric and fault detection. The second question recognizes the importance of event context in EDS test cases. By varying the amount of event context used in computing the metric, a finer grained measure of the similarity between test suites is garnered.

In setting up this study, several subject applications were chosen, test suites were developed and run, and the suites were compared based on several metrics. Each of these actions are described in the following sections.

3.1 Subject applications

Four popular, open source Java applications were chosen for this research and downloaded from SourceForge:

1. CrosswordSage 0.3.5[1], a popular tool for creating and solving professional-looking crossword puzzles with built-in word suggestion capabilities, with an all-time activity rate of 78.28%.
2. FreeMind 0.8.0[2], a very popular mind-mapping application, with an all-time activity rate of 100%.
3. GanttProject 2.0.1[3], a project scheduling application featuring Gantt chart, resource management, calendars, and the option to import/export MS Project, HTML, PDF, and spreadsheets, with an all-time activity rate of 99.98%.
4. jMSN 0.9.9b2[4], a clone of MSN Messenger, including instant messaging, file sharing, and additional chat features standard in MSN Messenger, with an all-time activity rate of 98.62%.

These applications were chosen for several reasons. All of the applications have an active developer community and high all-time-activity scores on SourceForge, with three of the applications above 90%. CrosswordSage was chosen partially because it is fairly new (first released in 2005) and yet has an activity score of almost 80%. These applications have also been released in several versions and have undergone quality assurance prior to each release.

3.2 Tools

The **GUI Testing FrAmewoRk (GUITAR)** was used to perform the study [15]. The **JavaGUIRipper**, one of the tools in the GUITAR suite, was used to glean the structure of the subject applications. By using Java Reflection, the JavaGUIRipper creates an XML file that represents the windows, menu items, and buttons present in the GUI, including the actions that are executed when those items are selected.

EFG-based test cases can be created using a **parameterized test case generator**, developed in previous work [21]. The EFG is annotated based on learned event interactions and the EFG is created. Test cases are then generated to exhaustively cover events up to *n*, given as a parameter to the generator.

Another tool in the GUITAR tool suite, the **JavaGUIReplayer**, was used for test case execution. The JavaGUIReplayer is a framework that opens the application under test and replays XML test cases containing details on the steps to be performed. Each event is executed on the GUI, and the state of the GUI is recorded after each step. The state is saved in XML files that can be examined to determine which test cases failed and why.

3.3 Test suites

Our empirical study used four test suites as the basis for determining the usefulness of the test suite similarity metric.

[1] http://sourceforge.net/projects/crosswordsage

[2] http://sourceforge.net/projects/freemind

[3] http://sourceforge.net/projects/ganttproject

[4] http://sourceforge.net/projects/jmsn

The suites consisted of: one model-generated suite (T_{orig}), which was executed to obtain its statement and method coverage; and three suites reduced from the model-generated suite by statement (T_{stmt}), method (T_{method}), and event pair (T_{pair}) coverage. The T_{orig} suite was generated from the EFG model of the GUI for each application.

Test suite sizes are as follows. For CrosswordSage, T_{orig} has 1903 test cases, T_{method} has 5 test cases, T_{stmt} has 11 test cases and T_{pair} has 854 test cases. FreeMind's T_{orig} has 58301 test cases; the method, statement and pair reduced suites have 135, 104, 18157 test cases, respectively. GanttProject's T_{orig} has 29133 test cases; T_{method} has 42 test cases; T_{stmt} has 88 test cases; and T_{pair} has 19656 test cases. JMSN's T_{orig} has 4634 test cases; the method, statement and pair reduced suites have 2, 19, and 3482 test cases, respectively. It is interesting to note the drastic size difference between the suites reduced on code coverage and the suites reduced on event pair coverage. We expect that this difference will have an impact on our results because the event-pair coverage reduced suites will be more similar to their respective original suites due to the added context opportunites for event execution.

3.4 Procedure

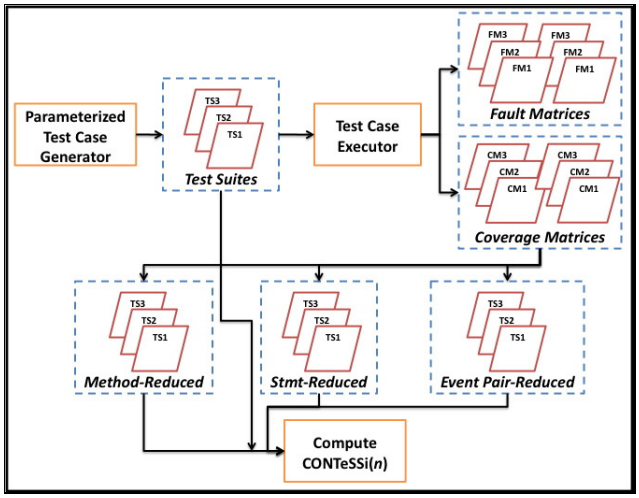

Figure 3. Comparing test suites using the CONTeSSi metric

Figure 3 gives a graphical representation of the steps described here. First, test suites based on the EFG were created using the parameterized test case generator. Next, the test suites were executed using GUITAR's JavaGUIReplayer. After running the test suites, fault detection and code coverage was collected for each test case, matrices were built, and this information was used to compare the suites. The test oracle used for this work detects crashes in these applications, where a crash is defined as an uncaught exception thrown during test case execution.

From the coverage matrices, reduced suites were obtained based on event pair, method and statement coverage. The code coverage and fault detection of these reduced suites was computed from the per-test-case coverage files generated during the execution of the original suite.

Finally, a file was created for each test suite where each line of the file represents one test case and contains the sequence of events in a test case. Using these files as input, $CONTeSSi(n)$ was computed, using values for n from 0 to 5, to compare the original suite to each reduced suite.

3.5 Comparing test suites

Using Equation 1, the $CONTeSSi$ metric is computed. To use this metric, we must also have a method of comparing suites with other metrics. The following function can be used to compare two suites given any of the metrics discussed here.

$$f(T_{orig}, T, m) = \frac{N(e_m(T_{orig}) \bigcap e_m(T))}{N(e_m(T_{orig}))} \qquad (2)$$

where T is the suite being compared to the original suite, m is one of the metrics on which suites are compared, such as statement coverage, branch coverage or event pair coverage, $e_m(suite)$ is a function returning the set of elements for metric m covered by $suite$, and N is a function returning the number in the set given. The result of this function is a number between 0 and 1 which represents the ratio of the number of $metric$ elements covered by both T and T_{orig} to the total number of $metric$ elements covered by T_{orig}. We will show coverage and fault detection numbers for each suite; then compare the suites based on these metrics.

3.6 Threats to validity

There are a few threats to validity which should be considered when interpreting the results of this study. First, due to our desire to use the existing GUITAR infrastructure, and to compare our results to those posted by previous graduate student researchers, we used subject applications developed in Java. Therefore, we have no information on how the results would translate to other development languages. Further, we are concerned only with EDS systems; this method may not be appropriate for test suites in other domains.

Second, although each application is different, they do not reflect all possible classes of EDS. Furthermore, the majority of the application code is written for the GUI, meaning the results may not be consistent for applications with a simple GUI and complex underlying business logic.

Another potential problem for this study is that it may not produce conclusive results on which value of n used for the context of $CONTeSSi(n)$ is most effective; however, it does give an indication of the impact of context and a trend of the results as the value of n is varied.

4. Results

First, the value returned by $CONTeSSi(n)$ is shown. Next, the traditional metrics (class, method, and block coverage) are shown. Finally, the two are compared. As the value of n increases for T_{pair}, we expect $CONTeSSi(n)$ to decrease in *most* cases, indicating a decrease in the similarity between the two suites as more context is considered, as expected from the example in Table 3. We do not have any expectations for T_{method} and T_{stmt} as these are not event-based reductions and therefore do not consider event context.

4.1 CONTeSSi vs. traditional metrics

Application	n	$Suite$ T_{method}	T_{pair}	T_{stmt}
CrosswordSage	0	0.788	0.974	0.760
	1	0.682	0.958	0.676
	2	0.597	0.952	0.614
	3	0.539	0.946	0.579
	4	0.496	0.942	0.552
	5	0.464	0.938	0.528
FreeMind	0	0.105	0.087	0.064
	1	0.088	0.079	0.051
	2	0.089	0.084	0.051
	3	0.091	0.089	0.162
	4	0.093	0.093	0.055
	5	0.095	0.098	0.057
GanttProject	0	0.200	0.947	0.358
	1	0.237	0.934	0.423
	2	0.229	0.933	0.414
	3	0.224	0.931	0.410
	4	0.220	0.930	0.408
	5	0.216	0.929	0.406
JMSN	0	0.233	0.612	0.491
	1	0.172	0.603	0.389
	2	0.146	0.615	0.355
	3	0.131	0.626	0.331
	4	0.121	0.635	0.312
	5	0.114	0.643	0.298

Table 4. CONTeSSi(n) for $T_{orig}, Suite$

Since we are interested in determining if the $CONTeSSi$ metric returns a value consistent with the "goodness" of a suite, we can use the information in Tables 4 and 5, combined with the fault detection reported in Table 7 to determine the most effective method of detecting test suite similarity. Table 4 shows the $CONTeSSi(n)$ value for each pair of test suites.

Application	Suite	Coverage Value Class	Method	Block
CrosswordSage	T_{orig}	41	20	25
	T_{method}	35	15	23
	T_{pair}	41	20	25
	T_{stmt}	35	20	25
FreeMind	T_{orig}	55	32	26
	T_{method}	51	29	24
	T_{pair}	55	32	26
	T_{stmt}	49	26	23
GanttProject	T_{orig}	66	51	46
	T_{method}	58	44	45
	T_{pair}	58	44	46
	T_{stmt}	66	50	45
JMSN	T_{orig}	35	24	27
	T_{Method}	28	16	20
	T_{Pair}	35	24	27
	T_{Stmt}	35	24	27

Table 5. Code coverage information

First, computing $CONTeSSi(n)$ without context (for $n = 0$), the data in Table 4 shows that the T_{pair} suites are the most similar to the original suite, T_{orig}, in three of the applications. For the fourth application, FreeMind, the most similar suite is T_{method}, followed by T_{pair} and T_{stmt}. In three of the four applications, T_{method} is more similar to T_{orig} than T_{stmt}.

Second, the relationships expected for T_{pair} are observed in Table 4. While there are some values of n that cause the $CONTeSSi(n)$ value to increase, the difference is so slight that it is unclear whether or not this result is significant.

Third, carrying the trends and relationships of Table 4 to fault detection (shown in Table 7) of each suite, it can be seen that the values returned by $CONTeSSi(n)$ are consistent with the faults detected by the suites. That is, for every application, T_{pair} detected almost the same faults as the original suite, while T_{method} and T_{stmt} detected fewer.

Fourth, traditional code coverage metrics are shown in Table 5. For all four applications, the class, method, and block coverage of the reduced suites are very similar to the original suite. Using this metric as a gauge for test suite similarity would lead a tester to believe the suites are very similar; however, the fault detection of each suite indicates otherwise. This finding supports the intuition described earlier that traditional metrics are not a good measure of similarity between test suites.

Application	Metric	Suite T_{method}	T_{pair}	T_{stmt}
CrosswordSage	method	1	1	1
	pair	0.016	1	0.022
	stmt	0.477	0.496	1
FreeMind	method	1	0.749	0.479
	pair	0.006	1	0.006
	stmt	0.974	0.759	1
GanttProject	method	1	0.999	0.968
	pair	0.003	1	0.003
	stmt	0.970	0.978	1
JMSN	method	1	1	0.200
	pair	0.347	1	0.008
	stmt	0.989	1	1

Table 6. Computing $f(T_{orig}, T, metric)$

Finally, Table 6 shows the computation of Equation 2 for each metric, for each application. Each combination of metric and test suite reduction method are compared, *i.e.*, the number of methods covered by the suites reduced by pair, method, and statement coverage are counted for each application. For almost every metric, T_{pair} covers the most elements of the metric. In most cases, the difference between T_{method} and T_{stmt} is very slight. Fault detection effectiveness of each suite further confirms this ranking of the suites.

4.2 Discussion

The similarity (or rather dissimilarity) between FreeMind's original suite and reduced suites does not follow the pattern of the other applications. This can be partially explained by the redundancy within test cases in the original suite, combined with the fact that the computation of $CONTeSSi(n)$ counts events (or event sequences). Because much of the redundancy is removed when the suites are reduced, the number of test cases as well as the counts of events (or event sequences) used in computing $CONTeSSi$ are much smaller. Additionally, these reduced suites did not find many faults; T_{pair}, however, had better fault detection than the others.

By comparing the test suites on existing metrics, which were also used to create the suites (Table 6), some insight into the value of these reduction techniques is gained. For all four applications, the similarity of T_{pair} to T_{orig}, measured in the ratio of elements covered, code coverage, and fault detection, also strengthens our claim that context is valuable in EDS test cases. This comparison also serves to reinforce the results provided by $CONTeSSi$ on the similarity and "goodness" of each suite.

$CONTeSSi(n)$ was designed with context in mind due to the importance of context in test cases for EDS. It is interesting to note the trend of the $CONTeSSi(n)$ value. In some suites for some of the applications, the value of $CONTeSSi(n+1)$ increases over $CONTeSSi(n)$ rather than decreasing as is the general overall trend. For example, the $CONTeSSi(1)$ value for GanttProject's T_{stmt} suite is larger than that of $CONTeSSi(0)$. The remaining $CONTeSSi$ values decrease, however, as n increases. Conversely, FreeMind and JMSN's T_{pair} values of $CONTeSSi$ for $n > 1$ increase as context is increased. It is possible this is due to the length of the test cases; as n gets closer to the length of the test case, the similarity between the suites increases.

5. Related work

Comparing whole test suites that are composed of sequences of events has not been researched in software testing. Some work has been done in the information retrieval (IR) and natural language processing (NLP) communities, however, in comparing large bodies of text. In IR, several researchers have developed similarity metrics to compare ranked lists that are output from an IR query, compare documents, and comparing a query to a document. Further, research on similarity between objects has been accomplished in software security to detect viruses and in neurocomputing to determine where to place an object in a fuzzy lattice. We will discuss these related works.

5.1 Similarity metrics in IR and NLP

Aslam and Frost developed a similarity metric for documents, as an extension of work by Dekang Lin in object similarity [2]. Lin *et al.*'s metric is designed to compare documents based on the features contained in that document, from some possible *feature set* which is contained in the set of documents. Aslam and Frost extend the metric to compare normalized documents, thereby accounting for fractional features that would otherwise be lost during normalization. They found their metric outperformed other standard metrics when run on a standard query retrieval data set.

Kilgarriff notes the problem of determining being able to come to a stable conclusion about a single word and that frequently, it is not possible to draw a conclusion about a combination of words or a rare word from a corpus [11]. In his work, Kilgarriff found that a method of comparison based on the χ^2 test is the most effective, followed closely by the Mann-Whitney ranks test [10]. He focused on determining both how similar two corpora are and in what ways two corpora differ, by determining which words are the most distinctive. Kilgarriff considered Poisson mixtures, Katz's model for word distributions, and adjusted frequencies used in research by Francis and Kučera. Finally, he used several

Applic.	Suite	Fault Id 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	Tot
Crossword Sage	T_{orig}	x	x	x	x	x	x	x															7
	T_{stmt}		x		x		x																3
	T_{method}		x																				1
	T_{pair}		x	x	x	x	x	x															6
Free Mind	T_{orig}	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	21
	T_{stmt}					x					x		x		x								4
	T_{method}								x														1
	T_{pair}	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x			x		x	17
Gantt Project	T_{orig}	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x						16
	T_{stmt}							x			x		x	x		x							5
	T_{method}			x																			1
	T_{pair}	x	x	x	x	x	x	x	x	x	x		x	x	x	x	x						15
JMSN	T_{orig}	x	x	x	x	x	x																6
	T_{stmt}			x	x																		2
	T_{method}																						0
	T_{pair}	x	x	x	x	x	x																6

Table 7. Faults detected in all applications

methods of comparing two corpora and determined several things: the distribution of word frequency is of little help; Spearman's rank correlation is too affected by the frequency of words that may not be important (such as *the*); and using a χ^2 test while ignoring the null hypothesis is a reasonable method of comparing corpora.

Previous research in IR, specifically the work by Aslam and Frost, is similar in concept to our interest in how similar test cases may be based on the events they contain, where *features* in information retrieval map to *events* in software testing. Kilgarriff's work in comparing document collections is very applicable to the work presented here. Just as is true for events in EDS test cases, Kilgarriff found words in a corpus do not appear in a random order and techniques to compare them must take this into account.

5.2 Similarity in software

Previous work in software engineering has compared *test cases* within a suite to determine redundancy inside the suite, but a suite to suite comparison has not been performed [13]. Traditional approaches for test suite minimization usually rely on coverage information collected during dynamic execution. Li *et al.*, applied a static analysis technique to detect redundant test cases based on their instruction sequences and counts.

Cosine similarity has been used in software engineering to detect code changes. Antonio *et al.*, used cosine similarity to determine the difference between classes in new versions of software, thereby pinpointing the refactored segments [1]. Karnik *et al.*, used the knowledge that malicious software shares significant amounts of code and the statistical properties of morphed viruses to successfully detect variants [8]. Traditionally, viruses are detected by examining method signatures for changes, but virus writers are aware of this approach and have methods of evading it. Although virus variants may differ in the sequence of instructions they contain, they are functionally the same. Karnik *et al.*'s method flagged any functions which are similar using a threshold value of 0.97, and then took the average of the similarities to evaluate overall program similarity.

Cripps and Nguyen proposed using cosine similarity measures as the inclusion measure used by fuzzy lattice neurocomputing (FLN) [5]. In this domain, data items of different types may be stored in the same lattice. Their work used counts of the *attributes* of each data item to generate a weighted vector which represents that item. These vectors can then be compared using the cosine similarity measure.

In software engineering, Karnik *et al.*'s work in virus detection is comparable if, instead of the machine instructions, events are used in the computations. The biggest difference is that in virus detection, events in a different sequence are functionally the same; this is not the case in EDS testing where event order is specific. We consider this in $CONTeSSi(n)$ by including the context in which the event was executed. Further, the approach of comparing each function to every other function differs from our comparison of whole vectors, which encode all events or event sequences in a suite.

6. Conclusions and future work

There are several existing techniques (*e.g.*, reduction and minimization) used to obtain test suites that are "similar" to an original suite. However, these techniques use criteria and metrics not suited to EDS. We presented a new parameterized metric called $CONTeSSi(n)$, which uses the context of n preceding events in test cases to quantify test suite similarity for EDS. $CONTeSSi(n)$ is appropriate for EDS because it considers the contextual relationships between

events. We defined and evaluated this metric on four test suites for four open source applications. Our results showed that $CONTeSSi(n)$ is a better indicator of the similarity of EDS test suites than existing metrics.

Our results have created several opportunities for future work. In the short term, we will extend our study to additional subjects to reduce threats to external validity. In the medium term, we will use $CONTeSSi(n)$ to develop a new reduction technique for GUI test suites. We expect that the reduced suite will be better at retaining the fault detection effectiveness of the original suite. We will also further investigate the relationship between the test case length and the value of n used in $CONTeSSi(n)$ to draw some conclusions on how to pick the best value of n, starting with a value of n which matches the length of the most test cases in the input suite. In the long term, we will apply $CONTeSSi(n)$ to other types of EDS, *e.g.*, web applications.

Acknowledgments

This work was partially supported by the US National Science Foundation under NSF grant CCF-0447864 and the Office of Naval Research grant N00014-05-1-0421.

References

[1] G. Antoniol, M. D. Penta, and E. Merlo. An automatic approach to identify class evolution discontinuities. *Principles of Software Evolution, International Workshop on*, 0:31–40, 2004.

[2] J. A. Aslam and M. Frost. An information-theoretic measure for document similarity. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 449–450, New York, NY, USA, 2003. ACM.

[3] V. Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, University of Maryland at College Park, 1992.

[4] P. A. Brooks and A. M. Memon. Automated GUI testing guided by usage profiles. In *ASE '07: Proc. of the 22nd IEEE/ACM Int'l Conf on Automated Software Engineering*, pages 333–342, New York, NY, USA, 2007. ACM.

[5] A. Cripps and N. Nguyen. Fuzzy lattice neurocomputing using weighted cosine similarity measure. *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pages 236–241, Aug. 2007.

[6] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.*, 2(3):270–285, 1993.

[7] J. Jones and M. Harrold. Test-suite reduction and prioritization for modified condition/decision coverage. *Software Engineering, IEEE Transactions on*, 29(3):195–209, March 2003.

[8] A. Karnik, S. Goswami, and R. Guha. Detecting obfuscated viruses using cosine similarity analysis. *Modelling & Simulation, 2007. AMS '07. First Asia International Conference on*, pages 165–170, March 2007.

[9] D. J. Kasik and H. G. George. Toward automatic generation of novice user test scripts. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 244–251, New York, NY, USA, 1996. ACM.

[10] A. Kilgarriff. Comparing corpora. *International Journal of Corpus Linguistics*, 6(1):1–37, 2001.

[11] A. Kilgarriff and G. Grefenstette. Introduction to the special issue on the web as corpus. *Comput. Linguist.*, 29(3):333–347, 2003.

[12] D. Leon and A. Podgurski. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*, pages 442–453, Nov. 2003.

[13] N. Li, P. Francis, and B. Robinson. Static detection of redundant test cases: An initial study. In *ISSRE*, pages 303–304, 2008.

[14] S. McMaster and A. Memon. Call-stack coverage for gui test suite reduction. *IEEE Transactions on Software Engineering*, 34(1):99–115, 2008.

[15] A. M. Memon and Q. Xie. Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. *IEEE Transactions on Software Engineering*, 31(10):884–896, 2005.

[16] X. Qu, M. B. Cohen, and K. Woolf. Combinatorial interaction regression testing: A study of test case generation and prioritization. *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 255–264, Oct. 2007.

[17] G. Rothermel, M. Harrold, J. Ostrin, and C. Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. *Software Maintenance, 1998. Proceedings. International Conference on*, pages 34–43, Nov 1998.

[18] Q. Xie and A. Memon. Automated model-based testing of community-driven open source GUI applications. In *Proc. 22nd Int'l Conf on Software Maintenance*, Sep 2006.

[19] T. R. Y and A. Kilgarriff. Measures for corpus similarity and homogeneity. In *Proceedings of the 3rd conference on Empirical Methods in Natural Language Processing*, pages 46–52. ACL-SIGDAT, 1998.

[20] E. Yilmaz, J. A. Aslam, and S. Robertson. A new rank correlation coefficient for information retrieval. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 587–594, New York, NY, USA, 2008. ACM.

[21] X. Yuan and A. M. Memon. Using GUI run-time state as feedback to generate test cases. In *ICSE'07, Proc. of the 29th Int'l Conf on Software Engineering*, Minneapolis, MN, USA, May 23–25, 2007.