Learning from User Workflows for the Characterization and Prediction of Software Crashes

Chloé Adam, Antoine Aliotti, Paul-Henry Cournède

Learning from User Workflows for the Characterization and Prediction of Software Crashes

Chloé ADAM[1,2], Antoine ALIOTTI[2], and Paul-Henry COURNÈDE[1]

[1] *Laboratory MICS, CentraleSupélec, 92290 Châtenay-Malabry, France*
[2] *GE Healthcare, 78530 Buc, France*

Abstract: Reducing as much as possible the rate of software crashes is crucial especially in medical applications. In this paper, we make the assumption that crashes result from the user workflow, that is to say the sequence of user actions. Our objective is thus to identify root causes of crashes and to anticipate them in real-time, based on the analysis of the sequences of user actions. For these purposes, we introduce two methods. The first one consists in using graph-based representations to detect combinations of user actions having a high probability to provoke a software crash, thus identifying crash signatures and helping for problem resolution. The second one, based on clustering of user sessions, is a real-time monitoring method, computing a crash probability at each new user action. Test cases show promising results for both methods. Our representation of user session as 'Graph-of-Actions' enabled the identification of some significant crash signatures while revealing the impact of successive actions dependence on crash causes. Likewise, our clustering based method for session monitoring resulted in promising values of sensitivity and specificity for some specific clustering configurations.

Keywords: Medical Applications; Software Crash; Graph-Of-Word; Graph-Of-Actions; Hierarchical Clustering.

1 Introduction

Radiologists use medical imaging solutions on a daily basis for diagnosis, surgery preparation or follow-up treatment, covering a large variety of pathologies. Today's imaging modalities (CT, MR, 3D X-ray, PET and PET/CT) present a huge challenge for the physician workflow: the number of patient exams increases while the time for exam data analysis and reporting decreases. Log files enable to record the evolution of the various software and system parameters during their use as well as the sequence of practitioner actions. A specific problem potentially occurring in the process is the crash of the analysis software

with severe consequences on the practitioner workflow. This is all the more true in the case of interventional applications, in which patient safety is put at stake. Therefore, reducing or annulling the crash rate of medical applications is a major line of the continuous improvement effort regarding the global quality and usability of medical software products. The aim of this study is to analyze the log file data for a better management of this crash risk. More specifically, we aim at identifying the root causes of the crashes and anticipate them in real-time.

Several factors such as user actions in the software interface (moves, selected tool, mouse clicks, etc.) or the system status (number of applications running, memory availability, dataset type, etc.) are likely to cause crashes. In this paper we will specifically focus on the user workflow. Thus, we will try to characterize and predict crashes based on the sequences of user actions. No real off-the-shelf solution currently exists, even though there are some comparable methodologies in other sequence learning tasks [1].

Numerous questions concerning the root causes of the crashes need to be addressed. Indeed, we do not know if crashes result from individual actions or from combinations of successive actions, if the order of actions has an impact... Similarly, we have no information about the exact moment in the session at which the crash is triggered. Answering these questions is even more challenging given the very high variability in the user sequences leading to crashes. To tackle these issues, the first objective of this paper is to propose a method enabling to detect patterns of user actions having a high crash probability. The motivation is here to help the software development teams enabling them to focus on specific program functions.

For this purpose, we will propose graph-based representations of the successive actions recorded in log files. We were greatly inspired by research works in the domain of information retrieval in text document. We used in particular the principle of the traditional bag-of-words filters, comparing relative frequencies of independent terms to detect spam emails or mobile messages [2]. We then moved to graph-based representations which have been widely investigated and applied in a large range of fields due to their powerful properties. Indeed, they enable to catch the direct dependencies between events. We based a part of our work on the representation used in [3], where user web sessions are transformed into weighted paths. To allow more flexibility to our representation, we also adapted the graph-of-words model proposed by [4] to our data.

The second objective of this paper is to introduce a real-time monitoring method, computing a crash probability at each new user action, based on the identification of clusters of user session queues (with the underlying assumption that the actions causing a crash are among the last ones before it happens). The motivation here is more related to risk management for the benefit of the users, imagining for example an automatic backup system triggered as soon as the crash probability exceeds a certain threshold. The research questions bear similarities with problems arising in the field of anomaly detection which is very used in the area of computer security, in particular for intrusion detection. A comparison of different approaches of anomaly detection was done in [5]. Several

similar methods were developed to detect intrusion using sequences of system calls. The principle is to build up a profile of normal behavior traces, treating deviations from this profile as anomalies [6, 7]. These techniques slightly vary in the definition of the dissimilarity measure used to compute how much a new sequence differs from existing normal sequences. Given the format of our data, we will in particular use the Hamming distance as in [6]. An interesting extrapolation of the same concept to detect network intrusions is based on clustering and classifies each new session as normal or as an attack depending on the dissimilarity measure between this session and the existing clusters [8]. Although not dedicated to anomaly detection, [9] is interesting since it uses a customized Levenshtein metric to cluster web sessions of variable lengths, this metric will also be tested for the application of our proposed method.

The paper first introduces a formal grammar enabling to translate the sequences of actions in log files into computational objects. We then propose in section III different representations enabling to detect combinations of user actions having a high probability to provoke crashes, and we compare the different representations on a test case. Section IV finally introduces our real-time monitoring method, computing a crash probability at each new user action, based on hierarchical clustering. Different configurations are tested.

2 Formalism

2.1 Data

In our application, a user session, which will be our reference measurement, corresponds to one patient images analysis by a physician. In other words, each session contains all the necessary mouse clicks to the radiologist to provide relevant medical recommendation to one patient. The sequence of user actions during this session is chronologically recorded in log files.

The dataset we used to test our both methods contains 5598 user sessions of variable lengths (minimal length of 3) and were obtained from identical hardware systems during a two-week radiology congress. 97 of them were prematurely interrupted by a software crash, that is, a crash rate amounting to 1.73%.

These crash sessions are characterized by a high variability in their sequences of actions and no obvious crash signature can be easily identified, making their characterization and prediction a difficult task. Figure 1 shows for each length of crash session, the total number of crash sessions (dark blue) and the number of different crash sequences (light blue). As can be seen, all the crash sessions being longer than or equal to 8 actions are different.

Moreover, regulations in medical applications impose many privacy restrictions regarding patient data, thus making the retrieval of this type of data even more difficult. Indeed, confidential data protection requires relatively complex and time-consuming procedures to ensure that the recovery process of log files is completely secure and that they provide no critical information.

The user mouse clicks recorded in log files actually refer to the corresponding

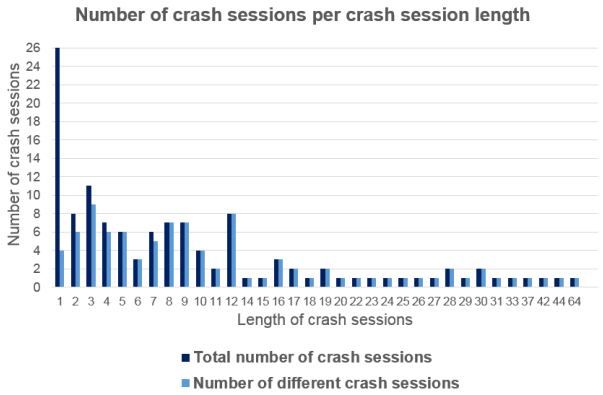

Figure 1: Number of different crash sessions per length.

function calls from the software source code. Therefore, the successive actions appear in the log files as strings of different lengths. To facilitate computations and analysis, we introduce a formal language to normalize actions representation into strings of the same length.

2.2 Formal Language and Definitions

Let Σ be an alphabet, that is to say a finite set of symbols, and D the dictionary assigning one user action to a symbol of Σ. Let X be a session of N actions, it will thus be represented as a word on the alphabet Σ, that is to say a finite sequence of N symbols in Σ, $X = x_1x_2...x_N$.

Let X be such sequence and consider a set of K symbols in Σ, $\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_K\}$. We will define below different types of inclusion relationships of the set of symbols Ω into the sequence X. It will prove useful to extract the significant actions (in terms of probable cause of crashes) from the sequence of actions.

Notations:

1. We will denote: $\{\alpha_1, \alpha_2, \ldots, \alpha_K\} \subset X$ if for all $1 \leq k \leq K$, $\alpha_k \in X - \{\alpha_1, \alpha_2, \ldots, \alpha_{k-1}\}$. It simply implies that all symbols in Ω belong to X, irrespective of the order. If $\alpha_i = \alpha_j = \alpha$ with $i \neq j$, and $\Omega \subset X$, then it means that α appears twice in X.

2. We will denote: $\alpha_1\alpha_2 \ldots \alpha_K \subset X$ if there exists i such that: $\alpha_1\alpha_2 \ldots \alpha_K = x_ix_{i+1} \ldots x_{i+K-1}$, that is to say that X contains the exact sequence $\alpha_1\alpha_2 \ldots \alpha_K$.

3. Finally, we will denote $[\alpha_1\alpha_2 \ldots \alpha_K]^W \subset X$ if there exists $i, i_1, i_2, \ldots, i_{K-1}$, with $i_k < W$ for all $1 \leq k \leq K-1$, such that: $\alpha_1\alpha_2 \ldots \alpha_K = x_ix_{i+i_1}x_{i+i_1+i_2} \ldots x_{i+i_1+i_2+\cdots+i_{K-1}}$. Here, it means that X contains the sequence $\alpha_1\alpha_2 \ldots \alpha_K$, but the consecutive terms α_i, α_{i+1} may be separated by up to $W-2$ symbols. For example, if $X = \alpha_1\beta\beta\alpha_2$,

$[\alpha_1, \alpha_2]^4 \subset X$. W is the maximal length of the "window" containing the symbols of interest.

In practice, considering the number of potential actions in the medical software of interest in this study, we choose without loss of generality to work with symbols composed of three letters, $\Sigma = \{\lll, \ggg, aaa, aba, ..., zzz\}$.

The symbol $\lll$ stands for the beginning of a session, thus for all sessions $x_1 = \lll$. Likewise, $\ggg$ stands for a clean exit of the software application, that is to say the end of a session without crash. All the other symbols represent actions.

With these notations we can easily distinguish normal sessions from crashes: a session of length N is a normal session when $x_N = \ggg$, while if $x_N \neq \ggg$, it is a crash. The set of normal sessions will be denoted $\mathcal{S}$ while $\mathcal{C}$ will represent the the set of crash sessions.

Beside, we will denote by Q_T the queue of length T of a session, corresponding to the T last symbols of a session different from $\ggg$.

Finally, for a finite set Ω, $|\Omega|$ will denote its cardinal.

3 Detection of Sequences of Actions with High Crash Probability

This section is dedicated to the detection of actions or sequences of actions which may cause the crash of a session with a high probability. For this purpose, we propose representations largely inspired by methods developed for information retrieval in text document. A review of the main graph-based representations of document is given in [10].

We will first compute the frequency of actions or groups of actions in crashes, with the underlying assumption that each action is independent from the others, as in the traditional *bag-of-words* model [2]. We will then use two graph-based representations, with elementary actions as graph nodes. The first one represents the sequences of actions as paths [3], the second one relies on the *graph-of-words* representation principle [4]. This last formalism allows capturing dependencies between actions while not being affected by potential transparent actions.

The computation of crash probability will remain the same in each of the following approaches, being simply equal to the number of times the studied actions or sequences of actions occur in crash sessions divided by the number of occurrences of these actions in both normal and crash sessions.

The three proposed representations will be described below. They will be illustrated on the specific case of the identification of couples of actions.

3.1 Representation 1 - *Independent Actions*

The first method consists in calculating the crash probability given a set of K independent actions $\{\alpha_1, \alpha_2, \ldots, \alpha_K\}$ in a user session X, as follows:

$$P_{Crash}(\{\alpha_1, \alpha_2, ..., \alpha_K\}) := P\left(X \in \mathcal{C} \mid \{\alpha_1, \alpha_2, \ldots, \alpha_K\} \subset X\right) = \frac{|\mathcal{C}_{\alpha_1,\alpha_2,...,\alpha_K}|}{|\mathcal{C}_{\alpha_1,\alpha_2,...,\alpha_K}| + |\mathcal{S}_{\alpha_1,\alpha_2,...,\alpha_K}|}$$

Where:

- $\mathcal{C}_{\alpha_1,\alpha_2,...,\alpha_K}$ denotes $\{X \in \mathcal{C} \mid \{\alpha_1, \alpha_2, \ldots, \alpha_K\} \subset X\}$ and thus $|\mathcal{C}_{\alpha_1,\alpha_2,...,\alpha_K}|$ is the number of crash sessions containing simultaneously all the symbols in $\{\alpha_1, \alpha_2, \ldots, \alpha_K\}$.

Likewise:

- $\mathcal{S}_{\alpha_1,\alpha_2,...,\alpha_K}$ denotes $\{X \in \mathcal{S} \mid \{\alpha_1, \alpha_2, \ldots, \alpha_K\} \subset X\}$ and $|\mathcal{S}_{\alpha_1,\alpha_2,...,\alpha_K}|$ is the number of normal sessions containing simultaneously all the symbols in $\{\alpha_1, \alpha_2, \ldots, \alpha_K\}$.

This representation does not take into account the connection between the different actions. This is probably not relevant, especially in the case of long user sessions. Indeed, the combination of actions occurring at the beginning and at the end of the session have probably no real interaction, contrary to actions occurring close to each other. This representation does not allow the distinction between the two cases.

3.2 Representation 2 - *Graph-of-Actions*

We propose a graph-based representation in which the dependence between user actions is captured via paths in the graph. We construct a directed and weighted graph from all the user sessions available in our database. Each directed edge uv will represent an occurrence of the sequence composed of the action u followed by the action v. We will use bidimensional weights, corresponding to $(|\mathcal{C}_{uv}|, |\mathcal{S}_{uv}|)$ where:

- $\mathcal{C}_{uv} := \{X \in \mathcal{C} \,|\, uv \subset X\}$ is the set of crash sessions containing the edge uv and $|\mathcal{C}_{uv}|$ is their number,
- $\mathcal{S}_{uv} := \{X \in \mathcal{S} \,|\, uv \subset X\}$ is the set of normal sessions containing the edge uv and $|\mathcal{S}_{uv}|$ is their number.

Each session will be successively added to the graph with the following process: the whole session is browsed, making an edge from every pair of successive actions. If the edge already exists, the bidimensional weight of the respective edge is updated, otherwise a new edge is created [3]. Edge occurrences in crash sessions will be distinguished from edge occurrences in normal sessions. The final graph vertices will represent all elementary actions available in the dataset. The principle of transformation of one user session into a path is illustrated in Figure 2.

This representation enables to capture proximity and order between actions.

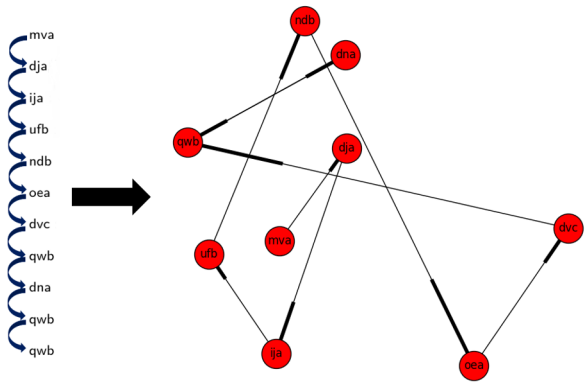

Figure 2: Graph construction from the example sequence of actions on the left side.

The aim is then to detect in the final graph, patterns or sequences of actions, which may lead to a high probability of crash. When we consider the detection of an ordered pair of actions, this problem simply corresponds to the computation of the crash probability per edge with the bidimensional weights as follows:

$$P_{Crash}(uv) = \frac{|\mathcal{C}_{uv}|}{|\mathcal{C}_{uv}| + |\mathcal{S}_{uv}|}$$

For sequences of length strictly greater than 2, the method is less direct since it is the probability of the path itself which needs to be computed and it can become very greedy from a computational point of view.

3.3 Representation 3 - *Graph-of-Actions with Sliding Window*

As there is a high variability in the sequences of actions leading to crash sessions, we make the assumption that there exist transparent actions in the sequences with no impact on crashes even if they appear in these sessions. To take this phenomenon into account, we take advantage of an idea proposed in [4] for graph-of-words in which they need to cope with meaningless words, in a situation very similar to our transparent actions.

The principle of construction of the graph is the same as in the previous method, except that one action will be linked by an edge to the $W - 1$ next following actions in the sequence, W being the length of the sliding window used to browse the user sessions. The Figure 3 shows an example of the construction principle with a sliding window of length equal to 3, meaning that each action will be directly linked to the first following action but also to the second one. These soft links enable to bypass eventual non influential actions that might skew the previous probability computations.

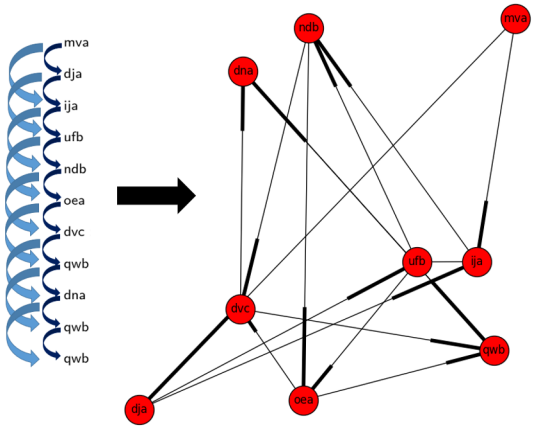

Figure 3: Graph construction with a sliding windows of length 3 from the example sequence of actions on the left side.

As in section 3.2, will use bidimensional weights adapted to this new graph, corresponding to $\left(\left|\mathcal{C}_{[uv]^W}\right|, \left|\mathcal{S}_{[uv]^W}\right|\right)$ where

- $\mathcal{C}_{[uv]^W} := \left\{X \in \mathcal{C} \,|\, [uv]^W \subset X\right\}$ is the set of crash sessions containing the 'soft' edge uv,
- $\mathcal{S}_{[uv]^W} := \left\{X \in \mathcal{S} \,|\, [uv]^W \subset X\right\}$ is the set of normal sessions containing the 'soft' edge uv.

The computation of the crash probability per edge thus adapts accordingly:

$$P_{Crash}([uv]^W) = \frac{\left|\mathcal{C}_{[uv]^W}\right|}{\left|\mathcal{C}_{[uv]^W}\right| + \left|\mathcal{S}_{[uv]^W}\right|}$$

3.4 Performance Evaluation

We test the different representations on the dataset introduced in Section 2.1. The aim is to detect pairs of actions having a high probability to cause a session crash, but also which occur several times, giving this way more weight and more significance to the results. The Table 1 provides for each method the number of detected couples of actions having a crash probability of 1 (N) and the number of times these couples occurred in crashes (O). Only the couples that occurred at least twice in crashes are recorded in the table.

The first method, *Independent Actions* enables to highlight 13 couples of actions, each of these couples occurred only twice and systematically the session in which they appeared ended with a crash. However, the very low number of occurrences prevents any significant analysis. In practice, the couple of actions thus determined did not help in understanding the causes of the crashes.

The second method, *Graph-of-Actions*, provides better results. 10 couples of actions among the 13 couples outlined by the first method are also detected,

Table 1: Detection of couples systematically leading to a session crash and occurring more than twice

	Number of couples N	Occurrences O
Independent Actions	13	2
Graph-of-Actions	10	2
	1	3
	1	**9**
Graph-of-Actions with Sliding Window	7	3
	1	4
	1	6
	1	**24**

but also occurring only twice. More interestingly, 2 additional couples are detected, one also occurring twice, but the other one occurring 9 times, which clearly becomes significant. Taking into account the dependence between action clearly improves the capacity of detection. The couple of actions when occurring independently appear more often in normal sessions, and consequently their associated crash probability is reduced with the first mode of representation.

The third method, *Graph-of-Actions with Sliding Window* enables to further improve the results: 7 couples occurring 3 times are detected. In addition, two other couples occurring respectively 4 and 6 times are detected. Finally, a pair of ordered actions leading 24 times to a crash is identified.

A crucial result is that the best detected pairs with the second and the third method, in bold in Table 4, actually correspond to the same couple of actions. This result provides precious information on the signature of the related crash. Let us note this couple $\{\alpha, \beta\}$, representing the succession of actions α and β in a user workflow. The second method shows that this combination when occurring, is systematically leading to a crash, it happened 9 times in this dataset. The third method, reveals that in 15 additional cases, the combination α - γ - β leads to a crash, γ being a non influential action.

For our specific application, two remarkable results were thus obtained: first, there is an impact of the combination of dependent actions to cause session crashes, and second there exist actions that are non-influential in terms of session crash and can be inserted in sequences of dependent (influential) actions without interfering. The representation of the sequences of actions as a graph with soft links obtained from a sliding window as in [4] seems a proper description of the software sessions for this purpose. A direct consequence was the identification of some significant crash signatures on our dataset, thus providing keys to the application developers to understand and solve the crash causes.

Beside this crash characterization, it would be particularly useful if we could provide a way to anticipate crash risks, in order for example to save automat-

ically the session, notify the user or force a clean exit of the application. The next session will propose such method, monitoring the crash risk resulting from the sequence of actions.

4 Real-Time Monitoring

4.1 Principle

The principle of our method is to use hierarchical clustering to define groups of sessions from our training dataset based on some similarity measure and to compute crash probabilities for the resulting clusters from the observations. Afterwards, during a working session, a crash probability is computed at each new user action based on the session assignment to the predefined clusters. Should this probability exceed a predetermined threshold, the current working session would be automatically saved and the user notified. The proposed method is largely inspired by [8], the main difference lies in the fact that we dispose of labeled sessions. We are thus able to compute a crash probability per cluster during the training step.

4.2 Hierarchical Session Clustering

To perform hierarchical clustering, three specifications have to be chosen:

- The metric that will be used to compute distances between observations,
- The linkage method,
- The dendrogram cutting height, determining the number of resulting clusters.

Each of these specifications will be detailed below and different configurations will be compared in the test section.

4.2.1 Metric

We make the assumption that crashes occur at the end of the sessions and we will cluster only the queues Q_T of the sessions (last T symbols of the session). To group the queues of the sessions into clusters, we need to compute a distance between the sequences of symbols contained in the queues. Considering our research problem and the remaining uncertainties, we have tested two existing measures of dissimilarity between strings (Hamming and Levenshtein) and a third metric that we have proposed (Intersection). These three measures are defined below, note that we will work with sequences of the same length.

- *Hamming*: the number of positions at which the corresponding symbols differ [6].

- *Levenshtein*: the minimum number of edition operations (insertions, deletions, substitutions) to transform one sequence of symbols into the other [9], [11].
- *Intersection*: the difference between the length of the studied sequences and the number of common symbols between both sequences.

Each of these metrics provides more or less flexibility regarding the position and the dependence of the user actions in the sequences. The Hamming measure imposes the strongest constraints in terms of similarity while the Intersection measure is the most permissive.

4.2.2 Linkage Method

Hierarchical clustering merges at each construction step the closest pair of clusters. Let G and H represent two clusters and $d_{ii'}$ the pairwise observation dissimilarities between an element i in cluster G and an element i' in cluster H. We recall the three classical main linkage methods to compute the dissimilarity $d(G, H)$ between G and H [12].

- Single linkage:
$$d_{Single}(G, H) = \min_{i \in G, i' \in H} d_{ii'}$$
- Average linkage:
$$d_{Average}(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{i' \in H} d_{ii'}$$

Where N_G and N_H are the respective number of observations in each group.

- Complete linkage:
$$d_{Complete}(G, H) = \max_{i \in G, i' \in H} d_{ii'}$$

4.2.3 Number of Clusters

Rather than computing the optimal number of clusters for each dendrogram, we have chosen to empirically determine the best cutting height for our problem.

4.3 Crash Probability per Cluster

For a given cutting height H of the dendrogram, we obtain K_H different clusters Ω_i of session queues, for $1 \neq i \neq K_H$. Each cluster Ω_i is partitioned into $\mathcal{C}_i$, corresponding to the crash sessions, and $\mathcal{S}_i$, for the normal sessions. The crash probability will be computed for each of these clusters, as follows:

$$P_{Crash}(\Omega_i) = \frac{|\mathcal{C}_i|}{|\mathcal{C}_i| + |\mathcal{S}_i|}$$

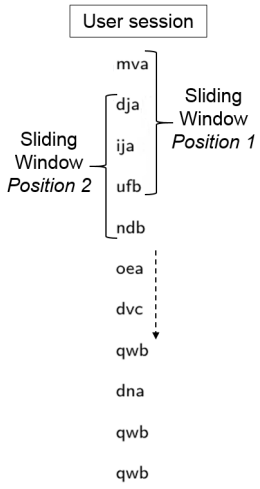

Figure 4: Real-Time Monitoring example on a user session

4.4 Crash Probability at each New User Action

The aim is then to go through an unlabeled session, crash or normal session, using a sliding window of length T, the same length as the clustered queues. At each step or each new user action, we will be able to assign the current session to a cluster and then derive a crash probability to the sequence contained in the sliding window based on the previous clustering. An example of the described process is given in Figure 4, using a sliding window of size 4.

At each new user action, a crash probability is computed for the sequence contained in the sliding window (corresponding to the last T actions of the current session). However, the corresponding sequence of T actions might be assigned to several clusters. These potential clusters may have different crash probabilities. Thus, we will consider the 3 following possibilities to assign the crash probability to the current session queue:

- *MinimumProba*: the smallest crash probability of all the potential clusters to which the sequence contained in the sliding window might be assigned.
- *MaximumProba*: the highest crash probability of all the potential clusters to which the sequence contained in the sliding window might be assigned.
- *AverageProba*: the average crash probability over all the potential clusters to which the sequence contained in the sliding window might be assigned.

4.5 Performance Evaluation

4.5.1 Crash Detection Threshold and Criteria

Let τ be the detection threshold, we list below the chosen classification criteria:

Table 2: Confusion Matrix

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

- An actual crash will be considered as detected if the crash probability value (Minimum, Maximum and Average will be tested) is above τ at least at one position of the sliding window during the whole monitoring of the user session.
- An actual normal session will be classified as a normal session if the crash probability value (Minimum, Maximum and Average will be tested) never exceeds τ.

4.5.2 Confusion Matrix

Given our binary problem, there are four possible outcomes (see Table 2):

- *True Negative*: an actual normal session which is predicted as normal.
- *True Positive*: an actual crash which is predicted as a crash.
- *False Negative*: an actual crash which is not detected.
- *False Positive*: an actual normal session which is detected as a crash.

4.5.3 Performance Metrics

Since we are in the case of highly imbalanced classes (around 2% of the sessions ended by a crash), we will use Specificity (also known as the *True Negative Rate*) and the Sensitivity (also known as the *True Positive Rate*), both defined below, to evaluate our method. Indeed, unlike the accuracy, these metrics do no depend on class distributions [13].

- $Specificity = \frac{TN}{TN+FP}$
- $Sensitivity = \frac{TP}{TP+FN}$

4.6 Results

4.6.1 Data Set Description

From the original dataset, we extract 1750 user sessions of length greater than or equal to 7, containing 33 crashes. The sessions were randomly split into a training set (80%), for the computation of the reference clusters, and a testing set (20%) composed of sessions on which we will test our method.

Table 3: Tested values of the specifications

Specifications	**Tested Values**
Metric	$Me = \{$Hamming, Levenshtein, Intersection$\}$
Linkage Method	$L = \{$Single, Average, Complete$\}$
Queues Length (Q_T)	$T = \{3, 4, 5, 6\}$
Cutting Height (D_M)	$M = \{2, 3\}$
Probability Score	$P = \{$Minimum, Average, Maximum$\}$
Detection Threshold	$\tau = 0.2$

4.6.2 Tested Configurations

As explained previously, three specifications have to be chosen for the hierarchical clustering (the metric, the linkage method and the cutting height) and two additional specifications relative to our method have to be defined (the length of the clustered queues and the probability score). All the possible configurations for each of these five specifications (described in Table 3) were tested, providing 216 different configurations of the type $\{Me\}\_\{L\}\_\{Q_T\}\_\{T_M\}\_\{P\}$.

Note that we decided to set the crash detection threshold to 0.2 to first determine the most promising configurations. Further work consists in varying this threshold value to build associated ROC curve. This representation will enable us to determine a threshold and a configuration providing the best trade-off between specificity and sensitivity given our initial problem.

4.6.3 Best Configurations Obtained

The best configurations we obtained are listed in the Table 4. Four of them, printed in bold, outperform the others. Three of them, marked with an asterisk (*), are very similar. Their specificities and sensitivities are both above 0.80. The fourth one, marked with two asterisks (**) only provides a sensitivity of 0.77, however it ensures a specificity value of 0.91. From the user perspective, this configuration is probably the best compromise between the true negative rate and the true positive rate, avoiding too many false alerts and still detecting 77% of the crashes.

Concerning the best specifications, the Levenshtein metric, an intermediate dissimilarity measure in terms of flexibility between Hamming and Intersection, appears in two of the four best configurations. The most appropriate linkage strategies seem to be the average and the complete linkage methods. The clustering of the queues of length equal to 6 and a cutting height of 2 provided the best performances, such as the choice of the minimum crash probability when assigning the sequence contained in the sliding window to a cluster.

Table 4: Real-Time Monitoring Best Configurations

Configurations	Specificity	Sensitivity
HAM_COMPLETE_Q6D3_MIN	0.80	0.79
HAM_COMPLETE_Q6D2_MIN	0.84	0.82
HAM_AVERAGE_Q6D3_MIN	0.76	0.81
HAM_AVERAGE_Q6D2_MIN*	**0.83**	**0.87**
HAM_AVERAGE_Q6D2_AVE	0.77	0.89
HAM_AVERAGE_Q5D2_MIN	0.80	0.81
HAM_AVERAGE_Q5D2_AVE	0.75	0.82
INT_AVERAGE_Q6D2_MIN**	**0.91**	**0.77**
INT_AVERAGE_Q6D2_AVE	0.84	0.81
LEV_COMPLETE_Q6D2_MIN*	**0.86**	**0.84**
LEV_COMPLETE_Q6D2_AVE	0.73	0.84
LEV_AVERAGE_Q6D3_MIN	0.80	0.79
LEV_AVERAGE_Q6D2_MIN*	**0.84**	**0.89**
LEV_AVERAGE_Q6D2_AVE	0.78	0.89
LEV_AVERAGE_Q5D2_MIN	0.80	0.79
LEV_AVERAGE_Q5D2_AVE	0.73	0.81

4.6.4 ROC Curves

The ROC curves were computed for the 4 best configurations (see Figure 5). Although the configuration (**) we initially identified as the most relevant one does not provide the largest area under curve, it is still the one that best fits our application needs: indeed, the proportion of crashes is very low (around 2 %), which means that having a high proportion of "false positive" predictions would end up in interrupting normal sessions far too often. Moreover, this configuration provides even better results with a detection threshold lowered to 0.125, increasing this way its sensitivity to 0.79 while maintaining a specificity of 0.91.

5 Conclusion

In this paper we proposed two methods. The first one aims at identifying the combination of actions with high probability to provoke crashes. The different representations tested for this purpose allowed us to conclude that there is an impact of the combination of successive actions to make sessions crash. Moreover, we showed that non-influential actions could be inserted in these sequences

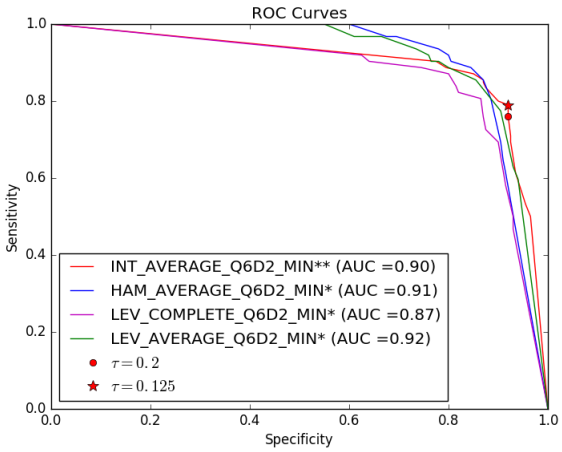

Figure 5: ROC curves for the 4 best configurations obtained.

of dependent actions without interfering in the probability of session crash and should thus be handled with in order to identify the proper crash signatures, otherwise we would miss the connection between the influential actions. Graphs with soft links to represent sequences of actions seem a proper description of the software user sessions for this purpose. Further work in this direction will concern the detection of longer paths.

The second method, real-time crash monitoring, provided promising results. Indeed we detected clustering configurations ensuring a specificity of 91% while still detecting 79% of the crashes. Another perspective of this work might be replacing the clustering of the session queues by the clustering of their graph-based representations based for example on the graph edit distance [14].

For both proposed approaches, we insisted on the methodology rather than on the exploitation of the outcomes. Indeed, a deeper analysis of the high crash probability couples detected in terms of software functions should be performed on a larger database of user sessions. This also applies for the determination of the optimal configuration in the case of the crash monitoring method based on clustering. However, it is important to note that data recovery can be quite complicated due to the regulations applied to medical devices.

Acknowledgment

We would like to thank the reviewers for their useful feed- backs which helped us to improve the article clarity.

References

[1] T. Lane and C. E. Brodley, "Temporal sequence learning and data reduction for anomaly detection," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 3, pp. 295–331, 1999.

[2] D. Sculley and G. M. Wachman, "Relaxed online svms for spam filtering," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 415–422.

[3] M. Eirinaki, M. Vazirgiannis, and D. Kapogiannis, "Web path recommendations based on page ranking and markov models," in *Proceedings of the 7th annual ACM international workshop on Web information and data management*. ACM, 2005, pp. 2–9.

[4] F. Rousseau and M. Vazirgiannis, "Graph-of-word and tw-idf: new approach to ad hoc ir," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 2013, pp. 59–68.

[5] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 1999, pp. 133–145.

[6] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, pp. 120–128.

[8] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*. Citeseer, 2001.

[9] A. Scherbina and S. Kuznetsov, "Clustering of web sessions using levenshtein metric," in *Industrial Conference on Data Mining*. Springer, 2004, pp. 127–133.

[10] R. Blanco and C. Lioma, "Graph-based term weighting for information retrieval," *Information retrieval*, vol. 15, no. 1, pp. 54–92, 2012.

[11] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, pp. 522–532, 1998.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning. Data Mining, Inference and Prediction*, Springer, Ed., 2009.

[13] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[14] B. Cao, Y. Li, and J. Yin, "Measuring similarity between graphs based on the levenshtein distance," *Appl. Math*, vol. 7, no. 1L, pp. 169–175, 2013.