Mobile Interaction Analysis: Towards a Novel Concept for Interaction Sequence Mining

Florian Lettner, Christian Grossauer, Clemens Holzmann
University of Applied Sciences Upper Austria
Department of Mobile Computing
Softwarepark 11, 4232 Hagenberg, Austria
firstname.lastname@fh-hagenberg.at

ABSTRACT

Identifying intentions of users when they launch an application on their smartphone, and understanding which tasks they actually execute, is a key problem in mobile usability analysis. First, knowing which tasks users actually execute is required for calculating common usability metrics such as task efficiency, error rates and effectiveness. Second, understanding how users perform these tasks is important for developers in order to validate designed interaction sequences for tasks (e.g. sequential steps required to successfully perform and complete a task). In this paper, we describe a novel approach for automatically extracting and grouping interaction sequences from users, assigning them to predefined tasks (e.g. writing an email) and visualising them in an intuitive way. Thus, we are able to find out if the designer's intention of how users should perform designed tasks, and how they actually execute them in the field, matches, and where it differs. This allows us to figure out if users find alternate ways of performing certain tasks, which contributes to the application design process. Moreover, if the users' perception of tasks differs from the designer's intention, we lay the foundation for recognising issues users may have while executing them.

Author Keywords

Usability task identification; pattern matching; navigation sequence visualisation;

ACM Classification Keywords

H.5.1. Information Interfaces and Presentation (e.g. HCI): User Interfaces – Evaluation/methodology

I.5.3.Pattern Recognition: Clustering – Similarity measures

General Terms

Human Factors; Design; Measurement.

MobileHCI '14, September 23 – 26 2014, Toronto, ON, Canada.

http://dx.doi.org/10.1145/2628363.2628384

INTRODUCTION

Associating user sequences (e.g. clickstreams) with actual tasks (e.g. writing an email) is of high interest for different fields, such as usability evaluation or e-commerce [19]. Although results for clickstream analysis of web logs (i.e. web farming) are available, there is little knowledge about task identification and interaction sequence mining of mobile applications.

In the course of a pilot study, we have interviewed developers and product managers of seven different companies from three different branches (i.e. automation engineering, m-commerce and telecommunications) about their requirements regarding effective and useful app analytics. It turned out that knowing what users intend to achieve when they use an app (i.e. which task they have in mind), and how they actually execute tasks that have been designed by developers, can be crucial in order to improve the user experience as well as the usability of mobile applications. For the companies we have interviewed, understanding how users execute tasks in different mobile contexts (e.g. activities) is essential for reliable usability studies.

In order to find out how users execute specific tasks, why they probably do not perform them as developers intended them to do, and which of them are more prominent and therefore executed more frequently than others, companies conduct expensive, time consuming controlled experiments to get feedback from end users. Additionally, they also resort to reviews and ratings from app stores. However, both ways have their shortcomings. The results of controlled studies are based on the limited observations examiners are able to make (e.g. figuring out the exact position where users tapped). Moreover, they are limited to a rather small number of participants. Results from the reviews and ratings are based on the subjective impression of users. For example, bad reviews often do not contain enough information about the real circumstances and what actually happened, and most of the users do not give feedback at all. Therefore, a lot of information about what users like, dislike or have problems with is unreachable for the developers.

To get access to objective data from real-world situations and obtain quantitative statements, the usage of an app can be tracked. A common method, which is used by companies like Microsoft for their products, is that users willingly sub-

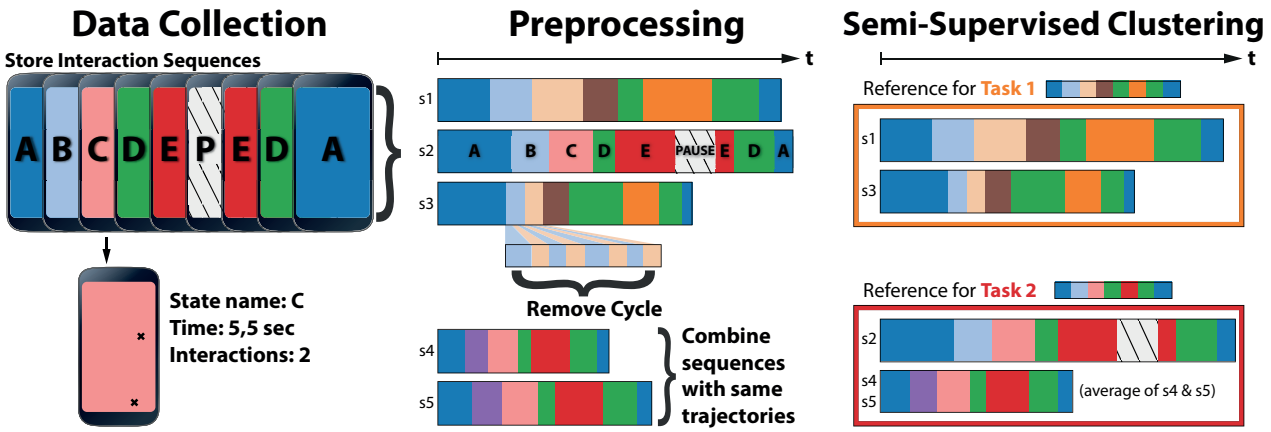

Figure 1. Illustrates the different stages of the proposed interaction sequence mining approach. First, user interaction sequences – i.e. ordered lists of UI states in mobile applications (e.g. page visits or dialogues popping up on a certain screen) triggered by user events (e.g. button clicks or long press gestures) – are collected from real users in the field. Second, the sequences are preprocessed by eliminating cycles and combining equal sequences. Finally, the sequences are grouped using a semi-supervised clustering approach and associated with reference tasks (e.g. writing an email).

mit their usage data (e.g. the Customer Experience Improvement Program for Microsoft Office). With this data available, developers can re-enact the actions of the users and can determine which task was executed.

Mapping tracked data to tasks for a large number of users is possible, but very time consuming if it needs to be done by a human. Based on logged interaction sequences, which contain information on visited states (e.g. a certain page), such as the amount of time that was spent in each state and the number of registered interactions (i.e. user events) in each state, we accomplished to automatically group interaction sequences based on their similarity (i.e. clustering) and to assign them to predefined reference tasks (i.e. classification), which is outlined in Fig. 1. Additionally, we describe in short how to visually prepare collected and clustered interaction sequences to allow designers to draw conclusions on their users' usage behaviour.

RELATED WORK

In web analytics, a lot of research has been done in the field of clickstream analysis. Web clickstream data is routinely collected to study how users browse the web or use a service. Shen et al. [19] point out that the ability to recognise and summarise user behaviour patterns from such data can be highly important for e-commerce companies in order to foresee potential user events (i.e. recommendation system) by logging how users click through a web page. Just like Zhang et al. [27], Shen et al. propose the usage of Markov Models in order to find the most probable path users will follow in accordance to previous user input, as well as the most probable user event at a certain time. Although Markov Models turn out to be very efficient for recommendation systems, we propose a different approach for interaction sequence mining, as we do not want to foresee possible future interaction sequences, but assign known interaction sequences to reference tasks to allow a comparison between the designers' perception of how users should perform a task and how users actually execute it in the field.

A different approach for clickstream analysis is presented by Ting et al. [23], who propose to use Continuous Common Subsequences in order to find unexpected browsing behaviours in clickstream data. Just like for the approach presented in this paper, expected routes have to be extracted beforehand. In contrast to the presented approach, Ting et al. transform these expected routes (i.e. common subsequences) into rules, which can be used to identify interaction sequences that do not match. Thus, this solution is only capable of finding browsing behaviours that do not match pre-defined rules. We, on the other hand, are able to identify and group interesting interaction sequences that do not completely match a specific reference task as well.

The most promising related approach is presented by Banerjee et al. [3], who try to associate user clickstreams on the web with certain tasks (i.e. buying a product on Amazon). Therefore, they propose to use a similar approach like the one presented in this paper. They record the time spent between clicks and use this information instead of the actual link trajectory for classification. As it turns out, classification solely based on time metrics is not sufficient in order to produce accurate assumptions regarding the belonging of clickstreams to a specific task. As a consequence, they manually associate hyperlinks and pages with semantic meanings (i.e. labels) and a probability beforehand. For example, it is very likely that users are in the process of buying a product, if they visit their shopping carts. In contrast to Banerjee et al., we do not need to associate a semantic meaning with a certain state or user event beforehand by introducing additional metrics.

Generally, clickstream analysis or web farming is important for predicting user behaviour such as personalizing websites, preloading content, creating recommendations or sorting search results [5, 6, 21, 10]. In contrast, we support task identification (i.e. clustering) to analyse and compare different interaction sequences collected from real users. Thus, we are able to find out which tasks users actually execute and how they differ from the developers' intention. This has been investigated by Coman et al. [8] in the context of au-

tomatically identifying different tasks of a software development session (e.g. clean, build, run). In case of Coman, task-related subsections as well as a rough approximation of their position in a development session have to be manually predefined. In contrast, the predefined tasks required for our semi-supervised clustering process can be recorded by actually using the application. Thus, it is possible to record new reference tasks on the fly without changing the application's source code.

For visualising the tracked data and the results of the analysis process, we examined visualisation techniques for finding patterns and interesting structures within sequences as well as for being able to compare several sequences. Church et al. introduced *Dotplots* which are used to find structures and compare similarities in text or source code [7]. Interesting structures can be found in form of diagonals, squares and texture within the dotplots. Wattenberg et al. [25] criticised that dotplots can get confusing when applied to strings with frequent repeated substrings, which would be a problem for our dataset as well. Wattenbergs' *Arc Diagrams* scale efficiently for strings with many repeating instances. Unfortunately for representing and comparing several sequences arc diagrams do not scale well.

Lastly, Sankey Diagrams were considered for displaying our results. As mentioned by Riehmann et al., Sankey diagrams are traditionally used for visualizing a flow of energy or material in networks and processes [18]. Riehmanns' *Interactive Sankey Diagrams* should facilitate the exploration of complex energy flows. Sankey diagrams support investigation of energy distribution, finding energy losses and the allocation of what energy is used for which purpose. These questions can easily be transferred to our field of displaying interaction sequences: interaction distribution, locating the places where users cancel an interaction sequence (i.e. task) and determining what interactions lead to the execution of a specific task. Therefore, we decided to represent the tracked data and the results of the analysis process with a Sankey based visualisation. As the presented visualisation is not focus of this paper, only an internal evaluation within our department has been performed by now.

SEQUENCE MINING

In order to extract user interaction sequences produced by real users in the field, and to assign them to predefined reference tasks, we propose a multi-stage process. The process consists of two preprocessing steps to speed up the clustering process on the one hand, and to make it more robust against errors on the other hand. Collected user interaction sequences are clustered by different similarity estimates, which are calculated between the collected sequences and the predefined reference tasks. For the calculation of these estimates, the sequence trajectories as well as the number of interactions and the amount of time spent in the different states are used. This section outlines how each part of the presented approach works.

Data Collection

User interaction sequences are collected for each session (i.e. between each start and stop of an application) and each user with the interaction logging framework presented in [13]. An interaction sequence basically consists of a sequential arrangement of states. The definition of states by annotating source code is mainly up to the developers' decision. However, in most situations, a state will be defined as a certain visual representation (e.g. page or screen), which is valid under a defined context. For example, one possible state in the Facebook app would be the *news feed* state. Although the screen appearance might change due to dynamic input (e.g. new status posts), the state will stay the same because this input is related to the same contextual information, namely status posts. Regarding data collection, each state is augmented with the time users spend actively in it, as well as the interactions (e.g. button clicks, text field input, gestures, etc.) that take place on the associated screen. External influences, such as interrupts by incoming voice calls or turning off the display, are recognised automatically and marked as *pause interval* states within interaction sequences (see Fig. 2).

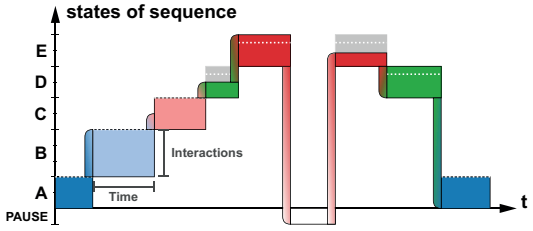

Figure 2. Illustrates a single interaction sequence. Different states are printed in different colors, their length indicates the time spent in these states, their height stands for the number of interactions that took place while the node has been active.

Preprocessing

Because interaction sequences are potentially collected by thousands of different users, the proposed solution requires certain preprocessing steps for the following reasons:

- It is very likely that common interaction sequences occur multiple times. Thus, equal interaction sequences (i.e. sequences that share the exact same trajectory) can be merged, in which case the logged user interactions and times in states are averaged to be more robust against outliers. As a result, we can assume that classification accuracy improves the more often a certain interaction sequence occurs. In addition, we are also capable of identifying obviously more important user sequences, which are probably related to a specific task and to separate them from random, rarely occurring interaction sequences.
- Interaction sequences can consist of sub-sequences that are repeated multiple times (e.g. switching back and forth between state A and B $\rightarrow$ ABABAB). We assume that repeating subsequences, which share the same trajectory, does not change the contextual meaning of a sequence. For example, no matter how often a specific setting in an app is changed, it refers to changing this specific setting (e.g. ABABAB $\equiv$ (AB)*).

Merging Sequences with Equal Trajectory

As mentioned above, merging interaction sequences that share the same trajectory (e.g. $a = b = ABC$), is done for

multiple reasons. First of all, it is possible to rank all collected user sequences according to their prominence (i.e. number of occurrences). Second, collected session times and user interactions, such as button clicks, can be averaged on the basis of datasets from many different users, which will make the sequence more robust against outliers for similarity estimation and clustering. Third, it will improve the runtime performance as it tremendously reduces the number of sequences that have to be compared to reference sequences in terms of their similarity.

Based on unique hash values for the trajectories of each interaction sequence, the list of all collected sequences can be sorted. As a result, all interaction sequences with the exact same trajectories, and thus sharing the same hash value, are grouped together. The count for each unique hash value can be determined by iterating over the entire list of interaction sequences.

Extracting and Substituting Cycles

Interaction sequences are chronologically ordered chains of states, such as page visits, or changes in the view hierarchy by interactions with widgets (e.g. dialog popup). By looking at them from a more abstract point of view, interaction sequences can be interpreted as character text, where each sequence represents a single word (e.g. $MainMenu = A$). Each single state (e.g. a page visit) can be seen as a single literal in the character text. As a result, it is possible to measure the similarity between two interaction sequences based on their string similarity, which is an essential and well-researched process in biomedical engineering and other fields [24].

By looking at collected interaction sequences of an Android app for the first time, we encountered that certain interaction subsequences can occur multiple times within the same sequence. Subsequences that do not add additional semantic or contextual information to the interaction sequence (e.g. changing the same setting over and over again) and which are repeated in immediate succession, represent cycles. As string similarity methods heavily depend on the trajectory of the strings to be compared, and because they are often based on the number of edit operations to change one sequence into another, cycles inside interaction sequences can have a negative input on the result quality of the presented approach [14, 16, 20]. However, as these cycles are just recurrences of one and the same interaction subsequence, finding repeating subsequences inside an interaction sequence can be broken down to locating specific character strings embedded in character text. This can be efficiently represented with regular expressions [22].

Baeza-Yates et al. presented an algorithm for searching regular expressions in sub-linear time with suffix trees [2]. In computer science, suffix trees are compressed digital trees, containing all suffixes of a given text as keys and their positions as values [11] (see Fig. 3). Suffix trees can also be used to find all occurrences z of the patterns $P_1 ... P_n$ of a total length m as substrings [12]. Although constructing a suffix tree can be memory intense for larger texts (e.g. news paper articles), we consider them appropriate in order to find and count the occurrence of each subsequence within a given interaction sequence, because interaction sequences consist of a rather small number of variables.

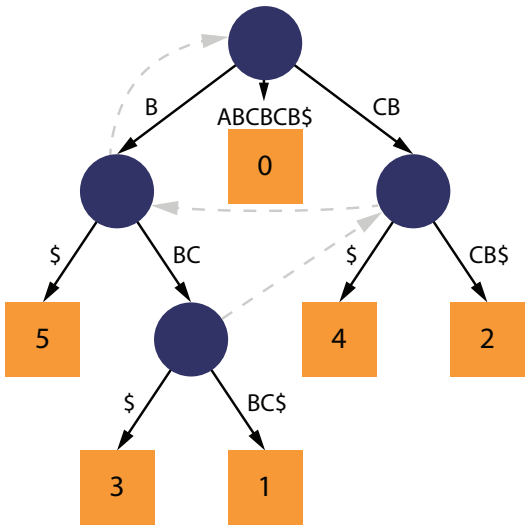

Figure 3. Simplified (i.e. compressed) suffix tree for the interaction sequence ABCBCB. Each subsequence is terminated by a terminal symbol (i.e. $). Each of the possible 6 suffixes (B$, CB$, BCB$, CBCB$, BCBCB$ and ABCBCB$) correspond to a path from the root node to a leaf, which are represented as boxes in the figure. The leaves store the actual start position of the suffix within the sequence. The dashed links are suffix links, which are used to construct the suffix tree.

By traversing the suffix tree, we can count how often a certain interaction subsequence appears inside the tree. Therefore, the algorithm counts how often a node (e.g. B, CB, BC, etc.) or subtree (e.g. BCB, CBCB, etc.) with a certain key is visited during the traversal. As a result, we can receive a list of interaction subsequences with their corresponding positions in the provided sequence, whose string length is greater than 1 and which appear more than once in the sequence.

In order to identify certain suffixes being cycled, the suffix positions, which are stored in leaves of the suffix tree, can be used. Thus, we iterate over all found suffix tree nodes and compare their positions based on the condition

$$s(i)_{idx} == s(i-1)_{idx} + len(s(i-1)) \qquad (1)$$

where $s(i)_{idx}$ is the start position of the current suffix and $len(s(i-1))$ the end position of the previous suffix within the sequence. It does not matter which of the subsequences is processed first, as the outcome will be the same for all possible situations (i.e. A(BC)*B ≡ ABCB and AB(CB)* ≡ ABCB). The information regarding node attributes, such as the number of interactions or session time, from eliminated cycles is not thrown away, but taken into account as averaged values for the regular expressions (i.e. BCBCBC ≡ (BC)*). In advance, cycle optimisation turns out to be a very promising preprocessing step, which will improve the result quality for the clustering process. To back these assumptions, we would like to refer to the results from our study as shown in Fig. 10, which demonstrates representatively for all metrics we investigated that the result quality is significantly higher if cycle optimization is applied.

Semi-Supervised Clustering of User Sequences

As outlined in the introduction of this section, clustering user interaction sequences by their contextual meaning (e.g. writing an email, buying a ticket, etc.) requires some sort of supervision to partition sequences, which share the same task, into one cluster, because it is not necessarily guaranteed that just the same classes of sequences (i.e. tasks) are grouped together. In contrast to Basu [4] and Finley et al. [9] however, the proposed approach does not require noticeable large training data sets to produce accurate results. For the presented approach, supervision can be reduced to predefining reference tasks, which partly turns the clustering problem into a classification problem [1]. These reference tasks can be created by app designers, because they think about and define how tasks are planned to be executed by users for their apps.

Similarity Estimation

In order to be able to decide which sequences should be clustered together and classified as a certain task, the similarity between these sequences and the reference tasks has to be determined. Therefore, we propose a combined similarity measure based on the three different and independent attributes (i) trajectory, (ii) time and (iii) number of interactions, which are used in the following in order to compute the similarity between reference tasks and user sequences.

String Similarity

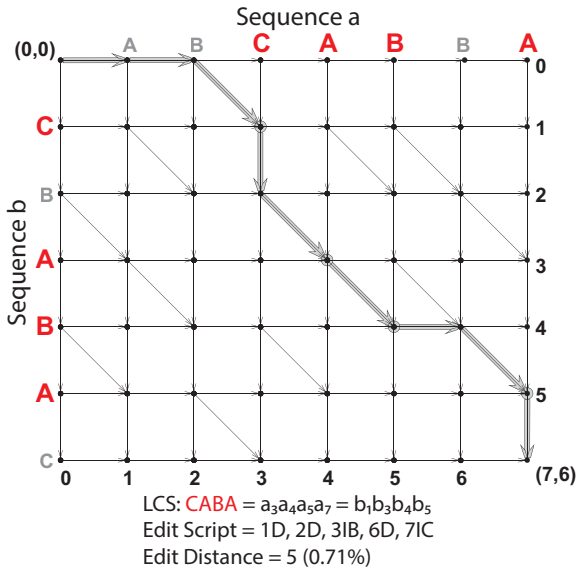

Figure 4. Edit graph illustrating the process of creating the edit script, and in the following the edit distance, for two different sequences. Source: [15]

.

As already mentioned, it is possible to treat interaction sequences as character text, where each state of the sequence is seen as a single literal. Finding the similarity based on the trajectory of a string is strongly related to finding DNA sequences in biomedical engineering.

One of the most common approaches there is to compare strings based on their *longest common subsequence (LCS)*. An LCS of two strings is a subsequence of maximal length that appears in both strings. The Levensthein (LV) distance (i.e. edit distance), which is defined as the number of editing steps (insert, delete or replace) required in passing from one string to the other, is a common measure for the similarity of two string based on their LCS [17].

Fig. 4 illustrates, how edit graphs can be used to calculate the similarity between two sequences based on their LCS. The edit script describes the number of edit operations required to transform one sequence (a) into another (b). The number of edit operations itself is defined as edit costs. In the graph, delete operations D are represented by horizontal steps, insert operations I are shown as vertical steps. A normalised score describing the similarity of the two paths can be described as

$$sim_{LV}(a,b) = 1 - \frac{n}{max(len_a, len_b)} = [0...1] \quad (2)$$

where n is the number of edit operations (i.e. insert, delete, replace) to change sequence a to sequence b and len_a and len_b are their lengths (i.e. total number of states). The following listing shows some simple examples for string similarities based on the edit distance:

$$sim_{LV}(ABC, ABC) = 1 - \frac{0}{max(3,3)} = 1.00 \quad (3)$$

$$sim_{LV}(ABC, ADC) = 1 - \frac{1}{max(3,3)} = 0.\overline{66} \quad (4)$$

$$sim_{LV}(ABC, AEF) = 1 - \frac{2}{max(3,3)} = 0.\overline{33} \quad (5)$$

One common problem with the Levensthein distance is that each edit operation is equally expensive. Unweighted edit costs can especially be a problem in long sequences, because mutations within a sequence can add too much noise to allow a meaningful comparison of those regions. As a result, the approach has to be modified in order to be able to obtain correct alignments in regions of low similarity between distantly related sequences. Algorithms to solve this issue have been proposed by Needleman et al. [16] and Smith et al. [20]. Both propose a solution for biological sequence comparison (i.e. protein and nucleotide sequences) based on dynamic programming. In contrast to Needleman, the algorithm proposed by Smith covers parts of the sequences and thus finds the best local alignment between two sequences instead of the best global alignment with respect to a scoring system (i.e. weight matrix) being used:

$$sim_{LV}(ABC, ABDEDFDC) = 0.37 \quad (6)$$

$$sim_{NW}(ABC, ABDEDFDC) = 0.56 \quad (7)$$

$$sim_{SW}(ABC, ABDEDFDC) = 0.\overline{66} \quad (8)$$

In contrast to the Levensthein distance (LV), Needleman-Wunsch (NW) as well as Smith-Waterman (SW) produce a much higher similarity score, as they are able to identify the subsequence $DEDFD$ as one linked delete operation. Thus, the edit costs for the whole substring are 1 instead of 5

Time Similarity

Although string similarity achieves good results per se, it can become unstable in case of shorter sequences (e.g. ABC and ABD). As all presented string similarity algorithms would return a similarity score of 0.66 for $sim(ABC, ABD)$, it is uncertain whether a single different state makes a new task out of it or not.

A different approach is presented by Banerjee et al. [3], who propose similarity estimation based on session times. As similarity measures solely based on time turn out to be too inaccurate for sequence clustering, we propose to use the similarity between sequences (a and b from different users) based on the averaged time T users spent in their LCS (with a length of n states) in addition to string similarities:

$$sim_{TIME}(a,b) = \frac{1}{n}\sum_{i=1}^{n}\frac{min(T^a_{LCS(i)}, T^b_{LCS(i)})}{max(T^a_{LCS(i)}, T^b_{LCS(i)})} \quad (9)$$

where $LCS(i)$ represents a single state in the LCS at position i and n is the number of total states in the LCS.

LCS Weight Estimation

In addition to similarity estimates, interaction sequences contain information that can be used in order to improve the reliability of the clustering process [3]. In this case, we will show how it can be used as additional, independent feature to enhance the result quality. Therefore, we propose to measure the impact or weight of the LCS of a sequence in relation to the entire sequence based on the time users spent in each state as well as the number of interactions that were executed within each state. As a consequence, it will be possible to rate the impact of an LCS. The longer the LCS, the more impact it has on the clustering process. The weight of the overlapping region of two sequences can be computed by

$$imp(a,b) = \sqrt{\frac{T^a_{LCS}}{T^a}\cdot\frac{T^b_{LCS}}{T^b}}\cdot\sqrt{\frac{I^a_{LCS}}{I^a}\cdot\frac{I^b_{LCS}}{I^b}} \quad (10)$$

where T refers to the total time, and I to the total number of interactions, users spent within a sequence (i.e. session time). T_{LCS} is the amount of time and I_{LCS} the number of interactions users spent in the LCS. As it is assumed that both sequences are not independent of each other (they share the same LCS), the geometric mean instead of the arithmetic mean is used in order to compute the average weight of both sequences.

We apply the same formula to the number of interactions to show how many interactions (e.g. button clicks) are performed in the LCS in relation to the total sequence. For example, the LCS for the two sequences $a = A_2B_5C_7D_4E_3F_5$ and $b = A_7C_5G_2H_3E_2F_2$, where the indices stand for the interactions in each state, is $ACEF$. Using the formula above, we get the following interaction weight:

$$imp(a,b) = \sqrt{\frac{17}{26}\cdot\frac{16}{21}} \approx 0.71 \quad (11)$$

Clustering and Classification

Based on the similarity and weight estimates, we compute a compound score representing the overall similarity between the collected user sequences and each predefined reference task. The single scores are treated as independent measures in the range of $[0...1]$ for a specific feature. Therefore, we can compute the compound score by plain multiplication of the similarity measures. The pair with the highest similarity is seen as the most-likely positive match. However, it cannot be guaranteed that a most-likely match always indicates the correct cluster. For example, users do not always perform a certain task when they interact with an application. Especially, when users start using an app, a learning phase, where they explore the application without a specific aim, needs to be taken into account. Moreover, it is possible that users coincidently identify new tasks, which have not been predefined by designers.

As a consequence, a threshold is required in order to allow to sort out user interaction sequences, which cannot be associated with a cluster unambiguously. This threshold will be used in order to eliminate sequences which would have been associated with a wrong cluster otherwise (i.e. false positives). However, thresholding is a tricky task, as an optimal threshold cannot be calculated automatically without using training data (i.e. supervised clustering). If the threshold is set too low, invalid sequences (e.g. random paths) will be added to the cluster with the highest similarity. If it is set too high, valid sequences, which otherwise would have been associated with the correct cluster, will be marked as "unknown" (i.e. false negative). Thus, the overall goal of the clustering process is to maximise the number of true positive results by minimizing the false positives at the same time. Based on the experiments we conducted for threshold estimation, we achieved good results with the following target function

$$f(x) = 0.7\cdot TP - 0.25\cdot FP - 0.05\cdot FN \quad (12)$$

where TP is the number of true positives, FP the number of false positives and FN the number of false negatives. The weight for false positives is also considered being higher, because a higher number of false positives will introduce noise to clusters.

SEQUENCE VISUALISATION

For displaying the tracked interaction sequences and the results of the semi-supervised clustering, we created a visualisation which enables app developers to take a look at the different ways how the users accomplished certain tasks. It is possible to compare different sequences, all associated with a certain task, solely based on their state visits or in combination with either the time spent or the number of interactions happened on each state. To visualise how users navigate through an app, we chose a visualisation based on Sankey diagrams, which is embedded in a web application (see Fig. 5, 6 and 7). The implementation of the visualisation builds on Kunal Bhallas'[1] d3[2] implementation of Sankey diagrams with the ability of displaying loop-backs. Fig. 5 shows 26 sequences assigned to one task, where each state (e.g. A, B or C) is represented as a (rectangular) node and each sequence

[1] `https://github.com/kunalb/`

[2] `http://d3js.org/`

(e.g. ABC, ABAC etc.) as the connecting paths through several nodes. The height of each node represents how often a state was activated/visited (e.g. A was visited 40 times). The width of a path connecting two states indicates how often the state at the end of the path was activated from the state at the beginning of the path.

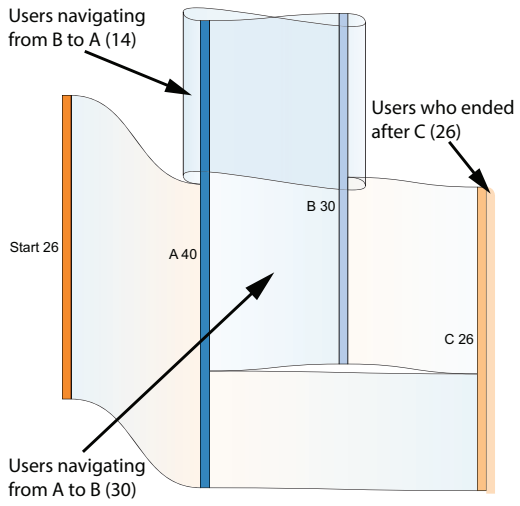

Figure 5. Visualisation of 26 sequences assigned to one task (predefined reference ABC) in different variations: AC, ABC, ABAC, ABABC.

Based on the nature of Sankey diagrams, the number of activations of a state (incoming paths, left side) has to be the same as the number of activations from a state (outgoing paths, right side). Inspired by Googles Flow Visualisation[3], we decided to add a path from the last node of the sequence leading nowhere, when a sequence ends (so called dropouts, as seen in Fig. 5 at the right side of node C).

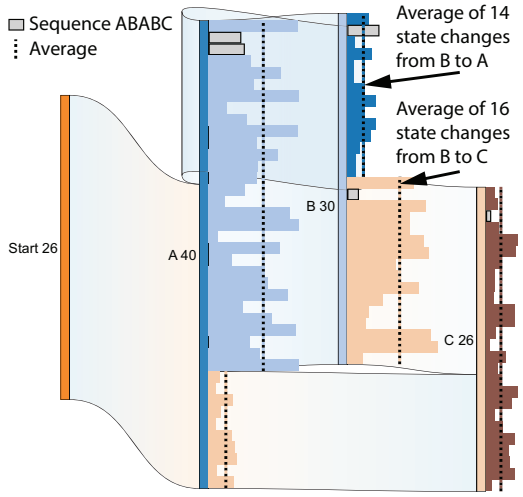

Figure 6. Visualisation of 26 sequences augmented with the time spent at each of the 96 state visits. By hovering over a state of a sequence (here ABABC), all corresponding states are highlighted. Additionally, the average times of the existing state changes are indicated by a vertical dashed line.

[3] `https://support.google.com/analytics/topic/2472754`

Additionally to the aggregated information of how often a certain node was visited and how the users navigated through the app, each single state visit is displayed as a horizontal bar next to its corresponding node. The length of the bar indicates either the number of interactions happened at this state or the time the user spent on it, depending on the viewers choice. Fig. 6 shows the same data as seen in Fig. 5 with all state visits represented as bars and one sequence (ABABC) highlighted. In this example, the bars are displayed in their chronological order.Vertical black dashed lines, where the intervals are filled with the same color as the target state, indicate the averaged current displayed values (in Fig. 6 the time spent at each state).

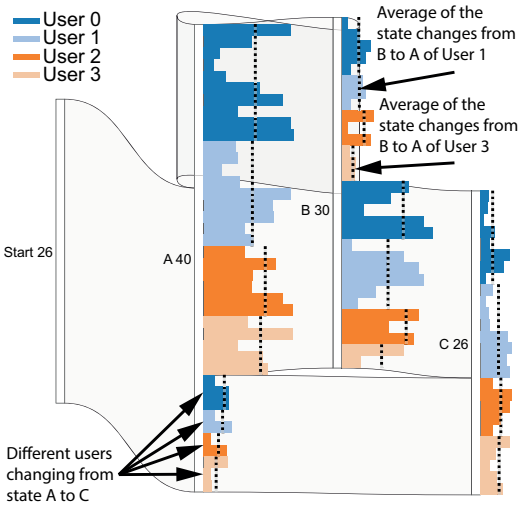

Figure 7. Visualisation of 26 sequences, grouped by their user and all executing the same task. Additionally, the average times of the existing state changes of each user are displayed as vertical dashed lines.

The viewer can change the representation from displaying common state changes (as seen in Fig. 5 and 6) to display grouped users or tasks. The size of the nodes and paths is not affected by this choice, only the arrangement of the horizontal bars. Fig. 7 again shows the same data as before, but with dissimilar users indicated by different colors of the relevant bars. The viewer has another two options when it comes to displaying the averaged values: First, as seen in Fig. 7, the average is obtained over each state change of the same user, second, it will be further separated between the different tasks (e.g. the average of User 0 changing from state A to B while performing Task 2).

EVALUATION

Participants

We conducted a study based on user interaction sequences, collected from real, unbiased users in the field, in order to find out how well the proposed clustering concept performs on user task classification. Therefore, we integrated the proposed data collection framework into an Android application for buying public transport tickets. Over a period of two weeks, we collected 500 interaction sequences from 150 arbitrarily selected, anonymous users, who were not instructed on how to use the application. The participants were notified and

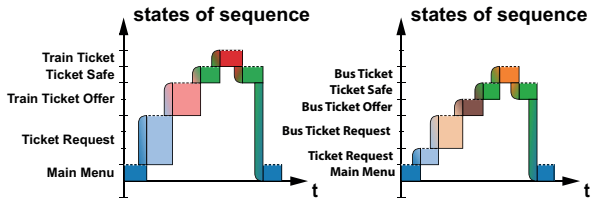

Figure 8. Predefined reference task for ordering a bus and a train ticket.

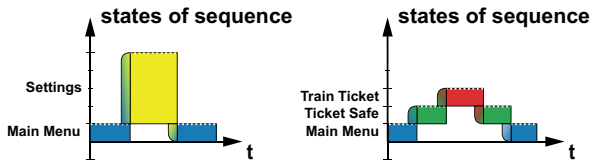

Figure 9. Predefined reference task for settings and verifying the ticket.

had to agree that user interaction data will be collected during application usage for scientific purposes. However they were not told which data exactly we collected and how it was processed.

Study Design

From these 500 collected interaction sequences, we extracted 200 unique paths by merging those sequences that share the exact same trajectories. For the experiment, we identified four different tasks, for which we recorded optimal interaction sequences each including time and number of interactions:

- **Order Train Ticket:** Users can order a train ticket. They have to select departure and destination station, number of persons (i.e. adults and children) and request an offer. Once an offer is received, the user can select either to pay the full ticket price or to get a discount based on a membership card. If the purchase order is confirmed, users get their ticket which will be displayed immediately. Alternatively, users can select to store a certain ticket request in their favourites. In this case, they can skip the ticket request procedure and will receive an offer directly.
- **Order Bus Ticket:** Users can order bus tickets for short-distance traffic in Austria's provincial capitals. Therefore, they have to select a city and the number of tickets they want to order. Additionally, they can choose between single and day tickets. After they received an offer, they can simply accept or refuse it. If users accept, they receive a short-distance traffic ticket, which will be displayed immediately.
- **Change Settings:** Users can predefine some settings for the ticket purchase process. They can for example predefine an id for memberships or select the payment service.
- **Verify Tickets:** Users can access a so-called ticket safe in order to have tickets verified by ticket inspectors. Therefore they can select any ticket (i.e. bus or train) from a list.

In order to verify clustering and classification results, we manually assigned the correct task labels (e.g. Order Bus Ticket) to all 500 interaction sequences as meta information. User sequences that we could not explicitly identify and associate with tasks (e.g. random, explorative paths) were labelled as "unknown", but were left in the test data set as noise to find out if they are sorted out accordingly by the algorithm.

Basically, user sequences – although executing a predefined task – might look quite different to reference sequences recorded by designers. For example, users might leave the application at any time (e.g. because of an incoming voice call). It is also very likely that they combine multiple tasks or that they repeat the same task multiple times. For example, they first change their settings and then order a ticket or they order a bus ticket after they ordered a train ticket. Thus, one of the main goals of this study is to show that the proposed solution is capable of dealing with such differences by ensuring that sequences that cannot be uniquely associated with a reference path are not added to a wrong cluster, or by applying preprocessing steps to increase the clustering robustness.

Results and Findings

The presented findings of the experiment are twofold. First, we were able to derive an optimal threshold for the provided test data set based on Equation 12 by repeating the clustering process for all possible thresholds in the range of 0.0 to 1.0 in steps of 0.05. Second, the accuracy for each possible combination of the proposed similarity measures was estimated for any threshold.

Threshold Determination

In order to provide accurate results for each possible combination of the proposed similarity measures, we determined the optimal threshold by exhaustively finding the optimal threshold for a manually classified subset.

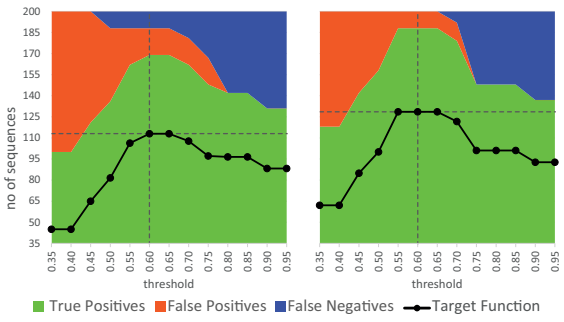

Figure 10. Clustering results with (right) and without (left) cycle optimisation based on string similarity only.

Fig. 10 shows the clustering result for the string similarity measure, where it is clearly visible that a better result (i.e. more true positives) can be achieved if cycle optimisation is used during preprocessing. In this case, the optimal threshold is 0.60, therefore Fig. 10 only shows a range around this values.

Similarity Estimation Accuracy

By determining the optimal threshold for each combination of the proposed similarity measures, we can calculate the accuracy for each of these combinations (see Table 1). In order

to show how the proposed measures perform under the worst case, cycle optimisation was not used during the preprocessing. The most promising related approach by Banerjee et al. [3], who used clickstream data from ordering articles in a webshop which is very similar to our ticketing application, achieved 66.67% of correctly clustered user sequences, however 33.33% of all sequences were associated with a wrong cluster. Banerjees' combined score only consists of the product of time similarity and time weight (see Table 1, orange row).

SS = String Similarity TS = Time Similarity
LWT = Time Weight LWI = Interaction Weight

SS	TS	LWT	LWI	TP	FP	FN
√	√	√	√	94.44%	0.00%	5.56%
√	√	√		72.22%	0.00%	27.78%
√	√		√	88.89%	0.00%	11.11%
√		√	√	77.78%	11.11%	11.11%
√		√	√	89.89%	0.00%	11.11%
√		√		83.33%	0.00%	16.67%
√			√	88.89%	5.56%	5.56%
√				88.89%	5.56%	5.56%
	√	√	√	83.33%	0.00%	16.67%
	√	√		66.67%	33.33%	0.00%
	√		√	77.78%	22.22%	0.00%
	√			50.00%	0.00%	50.00%
		√	√	88.89%	5.56%	5.56%
		√		88.89%	5.56%	5.56%
			√	83.33%	5.56%	11.11%

Table 1. Shows the clustering results for each possible combination of the proposed similarity measures, based on the optimal threshold for each combination (result of combined score of Banerjee et al. in orange).

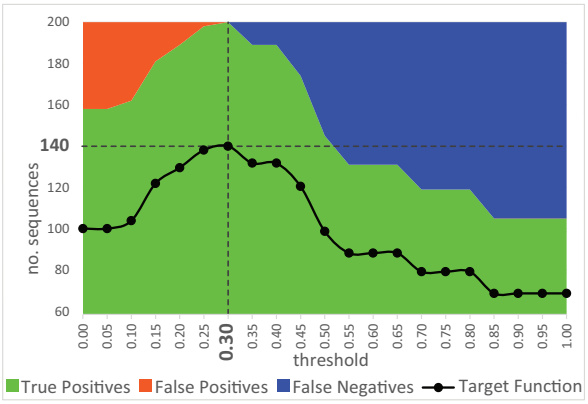

Figure 11. Clustering result by combining all four proposed similarity measures with cycle optimisation.

The best result is achieved by combining all four proposed similarity measures. In this case 94.44% of all sequences are clustered correctly. The remaining 5.56% of the sequences are labelled as "unknown" by mistake (i.e. false negatives). However, there are not any sequences that are associated with a wrong cluster (i.e. false positives). As shown in Fig. 11, by activating cycle optimisation as a preprocessing step, all user interaction sequences are clustered correctly.

CONCLUSION AND FUTURE WORK

In this paper, we presented a novel approach for user interaction sequence mining based on similarity measures that can be used in order to compare user-generated interaction sequences to predefined reference tasks. First, we showed how to preprocess collected user interaction sequences in order to increase the performance, and to make the clustering process more robust, by identifying and substituting cycles (i.e. repeated subsequences) within sequences. Second, we demonstrated how to compare collected interaction sequences with predefined reference tasks based on their trajectory (i.e. string similarity) and averaged session times. Additionally, we showed how further metrics (i.e. session time and interactions) can be used in order to calculate the weight of the longest common subsequence of interactions sequences sequence and their potential reference tasks.

In contrast to [3], we showed that it is not necessary to manually augment certain states with meta information such as category labels in order to correctly assign interaction sequences to the correct tasks.

Based on interaction sequences we collected from a ticketing application from real, unbiased users in the field, we showed that the presented approach works well for wizard type applications, where users are guided and the navigation the interaction flow (i.e. possible interaction sequences) is restricted. Based on the presented related work, which already achieved acceptable solutions in the web domain, we assume that the presented approach will also achieve valid results with other types of mobile applications (e.g. games). However, this is subject to future work, and currently work in progress.

In addition, we will further investigate thresholding in order to enhance the accuracy for applications with more degrees of freedom. Therefore, we will search for a possibility to include unsupervised feature weighting (i.e. Baysian feature weighting) such as proposed by Huang et al. [26].

Moreover, we are searching for a solution to get rid of the necessity for predefined tasks, which can probably be achieved with affinity propagation clustering.

Using the results of the clustering process, we are able to compare differences of how users execute one task. The presented visualisation supports the identification of interesting sequences (i.e. users who needed significant longer or triggered essential more interactions than the designer intended to execute a task), however still needs to be evaluated and compared with existing visualisations and analysis tools.

ACKNOWLEDGMENTS

The research presented is conducted within the Austrian project "AUToMAte – Automatic Usability Testing of Mobile Applications" funded by the Austrian Research Promotion Agency (FFG) under contract number 839094.

REFERENCES

1. Al-Harbi, S. H., and Rayward-Smith, V. J. Adapting k-means for supervised clustering. *Applied Intelligence 24*, 3 (June 2006), 219–226.

2. Baeza-Yates, R. A., and Gonnet, G. H. Fast text searching for regular expressions or automaton searching on tries. *J. ACM 43*, 6 (Nov. 1996), 915–936.
3. Banerjee, A., and Ghosh, J. Clickstream clustering using weighted longest common subsequences. In *In Proceedings of the Web Mining Workshop at the 1st SIAM Conference on Data Mining* (2001), 33–40.
4. Basu, S. *Semi-supervised Clustering: Probabilistic Models, Algorithms and Experiments*. PhD thesis, University of Texas at Austin, 2005.
5. Belkin, N. J. Helping people find what they don't know. *Commun. ACM 43*, 8 (Aug. 2000), 58–61.
6. Chi, E. H., Pirolli, P., and Pitkow, J. The scent of a site: A system for analyzing and predicting information scent, usage, and usability of a web site. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, ACM (New York, NY, USA, 2000), 161–168.
7. Church, K. W., and Helfman, J. I. Dotplot: A program for exploring self-similarity in millions of lines for text and code. *Journal of American Statistical Association, Institute for Mathematical Statistics and Interface Foundations of North America 2*, 2 (June 1993), 153–174.
8. Coman, I., and Sillitti, A. Automated identification of tasks in development sessions. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on* (June 2008), 212–217.
9. Finley, T., and Joachims, T. Supervised k-means clustering. Tech. rep., Department of Computer Science Cornell University, Ithaca, NY, USA, February 2008.
10. Fu, Y., Fu, H., and Au, P. An integration approach of data mining with web cache pre-fetching. In *Parallel and Distributed Processing and Applications*, J. Cao, L. Yang, M. Guo, and F. Lau, Eds., vol. 3358 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2005, 59–63.
11. Giegerich, R., and Kurtz, S. From ukkonen to mccreight and weiner: A unifying view of linear-time suffix tree construction. *Algorithmica 19* (1997), 331–353.
12. Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997, 123.
13. Lettner, F., and Holzmann, C. Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, MoMM '12, ACM (New York, NY, USA, 2012), 118–127.
14. Levenshtein, V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady 10* (1966), 707.
15. Myers, E. W. An O(ND) difference algorithm and its variations. *Algorithmica 1* (1986), 251–266.
16. Needleman, S. B., and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology 48*, 3 (Mar. 1970), 443–453.
17. Paterson, M., and Dank, V. Longest common subsequences. In *Mathematical Foundations of Computer Science 1994*, I. Prvara, B. Rovan, and P. Ruzika, Eds., vol. 841 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1994, 127–142.
18. Riehmann, P., Hanfler, M., and Froehlich, B. Interactive sankey diagrams. In *IEEE Symposium on Information Visualization. INFOVIS 2005.* (2005), 233–240.
19. Shen, Z., Wei, J., Ma, K.-L., and Sundaresan, N. Visual cluster exploration of web clickstream data. In *Proceedings of the 2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, VAST '12, IEEE Computer Society (Washington, DC, USA, 2012), 3–12.
20. Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of molecular biology 147*, 1 (Mar. 1981), 195–197.
21. Spiliopoulou, M. Web usage mining for web site evaluation. *Commun. ACM 43*, 8 (Aug. 2000), 127–134.
22. Thompson, K. Programming techniques: Regular expression search algorithm. *Commun. ACM 11*, 6 (June 1968), 419–422.
23. Ting, I.-H., Kimble, C., and Kudenko, D. Ubb mining: Finding unexpected browsing behaviour in clickstream data to improve a web site's design. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '05, IEEE Computer Society (Washington, DC, USA, 2005), 179–185.
24. Wang, J., Feng, J., and Li, G. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *Proc. VLDB Endow. 3*, 1-2 (Sept. 2010), 1219–1230.
25. Wattenberg, M. Arc diagrams: Visualizing structure in strings. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, INFOVIS '02, IEEE Computer Society (Washington, DC, USA, 2002), 110–116.
26. Yun, J., Jing, L., Yu, J., and Huang, H. Unsupervised feature weighting based on local feature relatedness. In *Advances in Knowledge Discovery and Data Mining*, J. Huang, L. Cao, and J. Srivastava, Eds., vol. 6634 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, 38–49.
27. Zhang, Z., and Nasraoui, O. Efficient hybrid web recommendations based on markov clickstream models and implicit search. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, WI '07, IEEE Computer Society (Washington, DC, USA, 2007), 621–627.