Data & Knowledge Engineering 117 (2018) 1–17

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Log sequence clustering for workflow mining in multi-workflow systems

Xumin Liu [a,*], Moayad Alshangiti [a], Chen Ding [b], Qi Yu [a]

[a] Golisano College of Computing and Information Science, Rochester Institute of Technology, USA
[b] Department of Computer Science, Ryerson University, Canada

ARTICLE INFO

Keywords:
Workflow mining
Sequence clustering
User behavior pattern
Probabilistic suffix tree
Non-negative matrix factorization

ABSTRACT

Current workflow mining efforts aim to discover process knowledge from user-system interaction logs and represent it as high-level workflow models. They assume there is one single workflow model in a system, or rely on the information that can explicitly link each log sequence to the underlying workflow model. Such assumptions may not be applicable to multi-workflow systems where the instances of different workflow models are mixed together without being differentiated. To address this issue, this paper proposes to apply sequence clustering methods to group similar log sequences together. Each sequence cluster corresponds to a workflow model and the log sequences in the cluster are the corresponding instances. This paper investigates different similarity measures, including structure-based and user-based, as well as different clustering algorithms, including one-side clustering and co-clustering. In order to incorporate user factors into sequence clustering, which is novel to the current sequence clustering methods, this paper proposes to model User Behavior Patterns (UBPs) as probabilistic distributions over sequences and learn it from the event log. We represent a UBP as a Probabilistic Suffix Tree and use it to measure sequence similarity. The co-clustering method leverages the dyad relationship between UBPs and log sequences to improve the clustering accuracy. An experimental study has been conducted and the result indicates that user-based methods outperform structure-based methods in terms of accuracy and they are more effective on dealing with noises in the log and the increase of log size. The UBP-sequence co-clustering method achieves the best performance which indicates the effectiveness of incorporating user factors and applying co-clustering.

1. Introduction

Workflow mining is to analyze event logs and construct a high-level representation of a system in the form of workflow models [33]. Different from a *de jure* model which is designed before the system is put in use, such a *de facto* model captures the actual behavior of a system and reveals how users interact with the system. It can be used as an important input for various analysis tasks, such as policy conformance check, error detection, and system diagnosis. As a result, workflow mining has attracted great attentions and enjoyed wide application domains over the past decade and its importance has been broadly acknowledged and emphasized in the areas of business process intelligence [11,24].

Traditional workflow mining approaches, such as the α-algorithm [31], usually learn one single model out of all the relevant event logs. This may generate a large, unstructured, and hard-to-understand workflow model (i.e., spaghetti model), from the analysis of

* Corresponding author.
E-mail address: xl@cs.rit.edu (X. Liu).

https://doi.org/10.1016/j.datak.2018.04.002
Received 5 August 2016; Received in revised form 23 March 2018; Accepted 3 April 2018
Available online 12 April 2018

event logs in those complex systems where there are diverse user interaction patterns [32] [36]. For example, in a medical insurance claim system, different types of patient treatment may go through various and diverse claiming processes. Spaghetti models are also common in Web-based systems where the logging is performed by a web server which is unable to capture the application-level workflow information. For example, an online store such as Amazon, supports different types of business processes which correspond to different types of user interactions, such as making purchases, selling products, requesting for refund/return, and maintaining baby registry.[1] All the actions for those processes are recorded into the same log without being specified what type of workflow they belong to. Learning one single model out of all the user interaction traces will result in a highly confusing spaghetti model, which does not contain much valuable knowledge about the latent workflow.

Current works have addressed the spaghetti model issue through clustering user interaction traces as a data pre-processing step for process mining [6,10,36]. By doing that, the traces belonging to the same workflow or sharing similar interaction pattern will be clustered together. Traditional workflow mining approaches can then be applied to learn a workflow model out of the traces within a cluster. These approaches mainly use a structure-based measurement to compute the similarity between sequences for clustering. This makes them not only sensible to noise in the logs, but also may fail to differentiate sequences belonging to different workflow models if these models are similar to some extent in terms of the operations they include and the execution order among those operations.

This paper is an extension of our previous work [20,21]. We propose to incorporate the user information into the clustering process. In particular, we target at those user-centric systems where only one user is involved in a workflow. Collaborative workflow models, where multiple users are involved in a workflow, will be considered in our future work. We believe that some hidden user factors, such as their preferences and requirements, may determine the way they use a system, which we coin as User Behavior Pattern (UBP). The UBP of a user can be reflected by the types of business processes, i.e., workflow models, the user has performed and the corresponding probabilities. For example, a frequent buyer's UBP would be very different from that of a frequent seller as the former usually purchases products while the latter frequently updates product inventories. As another example, websites like LinkedIn, provide different levels of services based on user membership types, i.e., regular users vs. commercial users. As a result, different users may have access to different workflow models. We expect that a user's UBP, which can be learned from the logs, can be leveraged to increase the sequence clustering accuracy. To further improve the accuracy of sequence clustering, we propose to leverage the dyad relationship between UBPs and sequences and co-cluster UBPs and sequences together. Co-clustering two correlated sets has been proved to more effective than one-way clustering [3,9], This will result in two types of clusters: *UBP clusters* that group users with the similar usage patterns and *sequence clusters* that discover the workflow models by grouping similar sequences. The two clustering processes leverage and improve each other's clustering result and hence co-evolve to reach better overall clustering performance. Co-clustering can also help solve process mining issues caused by process variants [18]. Users in the same cluster share a similar access pattern. Therefore, the instances of the same workflow tends to have the similar distance with the users in the cluster. Therefore, leveraging UBP clustering information to cluster sequences can help group those process variants, i.e., the instances of the same workflow, together. Overall, the proposed approach offers two key benefits for process mining over one-side clustering. First, taking into account the user factor incorporates additional relevant information besides sequences and the co-clustering process can help improve clustering accuracy. Second, through co-clustering, we also learn the knowledge related to users, i.e, discovering users with similar behavior patterns and the relationships between groups of users and workflow models.

The remainder of this paper is organized as follows. In Section 2, we describe an application scenario of a Web-based marketplace system. We use it as a running example to motivate and illustrate the proposed clustering methods. In Section 3, we present the clustering problem we address in this paper. In Section 4, we present an approach that clusters log sequences based on structural similarity. We present an integrated similarity measure to compare two log sequences and apply a revised k-means algorithm for the clustering. In Section 5, we present an approach that incorporates the user related factor, i.e., UBPs, to measure the distance between sequences. In Section 6, we present the measure for UBP-sequence distance and apply three-way matrix factorization onto the UBP-sequence distance matrix to co-cluster UBPs and sequences. In Section 7, we present an experimental study to demonstrate the effectiveness of the proposed approach. In Section 8, we discuss some representative related work. In Section 9, we conclude our paper and discuss future directions.

2. Case study: a web-based marketplace system

Suppose a Web-based marketplace system provides a platform for customers to buy and sell products. There are four types of users: administrator (e.g., employees), sellers, regular buyers, and window shoppers. Administrators manage and maintain the system, such as checking, removing, or updating customer records and product information. Sellers use the system to query and maintain product inventory, check orders, and arrange shipment. Regular buyers check product information and usually place an order after finding the desirable products. Window shoppers query and buy products, too. However, they tend to just look around more than actually buy stuffs.

Suppose there are multiple workflow models existing in this system, such as managing product information, managing customer information, purchasing products, requesting for return, and selling products. Fig. 1 shows two examples of such workflow models: customer management and product purchase. A user can carry out any workflow as long as he or she has the access. The user is classified based on his or her usage pattern, which is reflected as the frequencies of carrying out different workflows. Specifically,

[1] https://www.amazon.com/baby-reg.

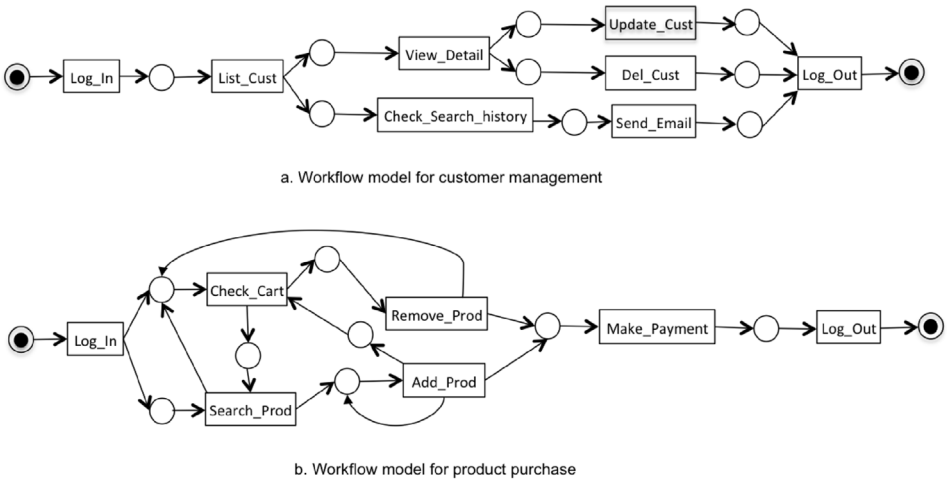

Fig. 1. Sample workflow models of the web-based marketplace system.

for administrators, they usually access five workflows, including product management, customer management, purchase processing, request processing, and order placement, with different probabilities, for example, 30%, 35%, 20%, 15% and 5%, accordingly. The grouping of users is completely determined by their behavior patterns. In case that there are two different administrators have totally different behavior patterns, they will be classified into different groups.

It is possible that different workflow models share some operations. For example, a user always needs to log into the system before making any update to the system. The operations log_in and log_out are included in all the related workflow models. Moreover, a user may check product details before making the order or canceling the order. The operations search_product is included in the three related workflow models. Meanwhile, due to the randomness of human behaviors, it is very likely that a user does not strictly follow the workflow design when using the system. For example, although the operation search_product is not included in the workflow model, an administrator may spontaneously search products when sending promotion emails to customers. Therefore, it is hard to differentiate log sequences that are the instances of different workflow models only based on the operations included in the sequences. It is beyond the scope of current process mining approaches to deal with such noises.

When a user enters into the Web-based system, he or she can click on any link or perform any operation that is available on the interactive interface. The invocation information recorded by the log server includes the session id (which is the same as case id), timestamp of the action, the operation id, the user identification (e.g., account name or IP address), and the duration of performing the operation. It is hard and even impossible to record the workflow model id in the log as neither the user nor the system has the enough knowledge or motivation of providing that information. Even in a system where workflow model id can be traced, it is still beneficial to ignore that information and learn it from the log so that the derived knowledge is more authentic and consistent with how the system actually behaves.

3. Problem statement

In this section, we first describe the content of an event log record and a sequence database. We then explain the problem of clustering log sequences addressed by this paper. Our approach takes the user-system interaction log data generated by a logging system as input. As existing logging systems usually have their own principles and designs, they support a variety of log file formats. Regardless of their differences, we can always find the following information in the log, including session id, operation id, user id, and timestamp. As we stated earlier, we assume that there is only one user involved in a session. Dealing with the collaborative workflow models is beyond the scope of this paper.

Table 1 shows a snippet of event log. We group the log records based on sessions. We do not care about the exact time when each operation is performed here. We use the timestamp to determine the temporal order between operations in a session and generate the corresponding log sequence, as shown in Table 2.

A log sequence can be formally specified as a tuple $l = \{uid, sid, s\}$, where uid is the id of a user, sid is the session id, and s is a sequence of operations performed, i.e., $s = s[1]s[2]\ldots s[n]$, where $s[i]$ is performed before $s[j]$ if $i < j$.

We propose a solution to learn the hidden links between log sequences and unknown workflow models. The input of this process is a sequence database, which stores a collection of log sequences. We model this problem as a clustering problem. More specifically, given a collection of session records, $\mathcal{S} = \{s_1, \ldots, s_n\}$, a vocabulary of operations, $\mathcal{OP} = \{op_1, \ldots, op_m\}$, and the number of workflow models, K, we will generate K sequence clusters, where similar sequences are grouped together. In this work, we assume that the number K is a prior knowledge obtained from domain experts. However, if K is not known beforehand, then we suggest using the current cluster number detection methods, such as the ones based on Gaussian Mixture Model (GMM) [29] and the ones based on gap detection [37] to derive K. Sequences in the same cluster are considered as the instances of the same workflow model. The output of this work can be served as the input of current workflow mining tool, such as ProM [35], to construct the Petri-Net representation.

Table 1
A snippet of event log.

session id	operation id	user id	timestamp
s_1	a	u_1	2013-03-08 14:59:30
s_1	c	u_1	2013-03-08 15:01:10
s_2	b	u_2	2013-03-08 15:01:12
s_2	c	u_2	2013-03-08 15:01:17
s_1	d	u_1	2013-03-08 15:02:09
s_3	a	u_3	2013-03-08 15:03:12
s_1	h	u_1	2013-03-08 15:04:11
s_2	d	u_2	2013-03-08 15:04:12
s_3	b	u_3	2013-03-08 15:05:12
s_4	a	u_1	2013-03-08 15:05:42
s_3	c	u_3	2013-03-08 15:05:53
s_5	a	u_2	2013-03-08 15:07:14
s_4	b	u_1	2013-03-08 15:07:17
s_5	b	u_2	2013-03-08 15:08:15
s_3	f	u_3	2013-03-08 15:08:22
s_5	c	u_2	2013-03-08 15:09:02
s_5	f	u_2	2013-03-08 15:10:09
s_5	a	u_2	2013-03-08 15:11:12

Table 2
The generated sequence database.

session id (*sid*)	user id (*uid*)	Operation Sequence (s)
s_1	u_1	*acdh*
s_2	u_2	*bcd*
s_3	u_3	*acf*
s_4	u_1	*ab*
s_5	u_2	*abcfa*
s_6	u_3	*abcf*

4. Structure-based clustering method

In this section, we present a clustering approach that compares sequences based on their structures. A log sequence can be treated as a string over operation vocabulary. Current string comparison methods can be applied to measure the similarity between log sequences, such as Levenshtein distance, Jaccard index, most frequent k characters, and overlap coefficient.

4.1. An integrated structure similarity measure

Algorithm 1 Measuring Similarity of Two Log Sequences
Input: log sequences s_i, s_j, two weights w_1, w_2
Output: Similarity $\lambda_{i,j}$
1: $d_1 = \frac{|s_i \cap s_j|}{|s_i \cup s_j|}$ {computing the similarity using Jaccard index}
2: **for all** $0 \leq t < |s_i|$ **do**
3: **if** $s_i[t] \notin s_j$ **then**
4: $s_i[t]$ is removed from s_i {Removing those private operations}
5: **end if**
6: **end for**
7: **for all** $0 \leq t < |s_j|$ **do**
8: **if** $s_j[t] \notin s_i$ **then**
9: $s_j[t]$ is removed from s_j {Removing those private operations}
10: **end if**
11: **end for**
12: $d_2 = 1 - \frac{(Dis(s_i, s_j, |s_i|, |s_j|))}{\max_{|s_i|,|s_j|}}$ {Compute edit distance and normalize it}
13: $\lambda_{i,j} = w_1 \times d_1 + w_2 \times d_2$

Putting sequence comparison in the context of workflow mining, Levenshtein distance, which is also known as edit distance, should be considered since it not only considers the operations included in the sequences, but also the order between operations. The similarity is measured based on the minimum number of steps for transforming a sequence to another sequence. Allowed transformations include inserting a new operation, removing an operation, and replacing an operation. However, it is not flexible enough to deal with parallelism, iteration, and possible noises in the log. Therefore, we propose to compare log sequence by integrating edit distance

with Jaccard index. Instances of the same workflow model tend to have the shared operations, i.e., the ones in common. Therefore, the shared operations between two sequences should contribute to their similarity, regardless of the order in the sequences. Private operations tend to imply different workflow models as they are unique to the models. They should contribute to the dissimilarity of two sequences. To avoid repeatedly punishing the similarity due to the private operations, those operations should not be considered when measuring edit distance. Furthermore, the distance needs to be normalized in the end.

Following the idea, we design the log sequence comparison, which is described in Algorithm 1. We first compare two sequences using Jaccard index, i.e., checking the ratio of the shared operations to all operations in the sequences (Line 1). The second part of the comparison is to apply edit distance, where the order of the operations is examined. Before computing the edit distance, the two sequences are revised by removing the private operations since the effect of those operations' existence on similarity measure is already counted and should not be considered again (Lines 2–11). After computing the edit distance between two revised sequences, we integrate the two distances computed using Jaccard index and edit distance using two weights, w_1 and w_2, where $w_1 + w_2 = 1$. w_1 indicates how important the number of common operations is and w_2 indicates how important the order of operation is to the similarity measure.

For example, let the two event sequences be 'abcdef' and 'acfdg'. The distance between these two sequences will be calculated as follows. The private operations between the sequences are 'b', 'e', and 'g'. So the jaccard distance will be $4/7 = 0.57$. After removing the private operations, the two sequences are revised as 'acdf' and 'acfd'. The edit distance between these new sequences is calculated as 2. 'acdf' can be transformed to 'acfd' by removing 'd' and adding 'd' after 'f'. This distance is normalized by the maximal length of the revised sequences. So the similarity is $1–2/4 = 0.5$. Suppose the two similarity measures are equally weighted. The final similarity is measured as $0.5 \times 0.57 + 0.5 \times 0.5 = 0.54$.

Complexity Analysis: The time complexity of computing the similarity using Jaccard index between two sequences, s_i and s_j, is $O(|s_i| + |s_j|)$ if a HashMap is used to check shared operations. Therefore, the complexity of computing all pairs of operations is $O(|S|^2 * l)$, where $|S|$ is the number of total sequences and l is the average length of sequences. The time complexity of computing the similarity using edit distance between two sequences, s_i and s_j, is $O(|s_i|*|s_j|)$. Therefore, the time complexity of computing all pairs of operations is $O(|S|^2 * l'^2)$, where $|S|$ is the number of total sequences and l' is the average length of sequences after removing the private operations during the comparison.

4.2. Sequence clustering

Using Algorithm 1, we can generate a similarity matrix $\mathcal{M}_s$ for log sequences by computing the similarity for each pair of sequences. $\mathcal{M}_s[i,j]$ is the similarity of the i-th sequence and j-th sequence. Using this matrix, we can apply data clustering methods to cluster the sequences. K-Means is one of the most widely used clustering method since it can efficiently generate clusters with a high accuracy. However, we cannot directly use it here since the clustering process depends on the centroids discovered during each iteration, where Euclidean distance is usually used. This is not the case in sequence clustering since no meaningful centroid can be found. Therefore, we need to revise K-Means to address this issue as follows.

We represent each sequence as a sequence vector with $|S|$ entries, i.e., $s_i = [sim_{i,0}, sim_{i,2}, \ldots, sim_{i,|S|}]$, where $sim_{i,j}$ is the similarity between the i-th and j-th sequences. Using this new representation, the similarity between two sequences can be evaluated using the cosine similarity of the corresponding vectors. Two sequences are similar if they have similar distances to all sequences, which results in a high cosine value. The similarity vector representation also allows computing the centroid of each cluster. Therefore, we can apply K-means clustering to these vectors to cluster sequences.

5. User-based clustering method

In this section, we present a clustering approach that compares sequences based on their inherent relationship with users. As we stated before, there are some user-related factors that could affect the user to use the system, i.e., the probability of carrying out each workflow model. The probability can be derived from the event log and then used to determine if two sequences are similar, i.e., the probability of them being the instances of the same workflow model. We propose to model a user behavior pattern, UBP, as a probabilistic distribution over sequences and then model each sequence as a UBP vector. We then compare two sequences using their vectors and apply data clustering methods to cluster sequences.

5.1. User behavior pattern model

In order to make our approach applicable to the most cases, we make the minimum assumption on the information we can get related to users, i.e., only the identification of a user, which can be account username, IP address, or MAC address. There can be other information, such as user's gender, age, education level, etc, that can be leveraged to help learn how users use the system. However, such information may not be obtained in the log server. By using the user id, we can generate an ordered sequence set, $\widehat{S}^i$ for each user i, which contains all the sequences in the log records generated by the user. For example, the sequence set for user u_1 is, $\widehat{S}^1 = \{acdh, ab\}$.

Among various probabilistic modeling methodologies, Hidden Markov Model (HMM) is widely used to model sequences as a Markov process with hidden states [5]. It can effectively capture the sequence patterns from observations on operation occurrence and predict further events. Considering the commonly massive size of log data, we propose to model a UBP as a **Probabilistic Suffix Tree** (PST) [1,26], a probabilistic distribution model of sequences, generated by the sequences contained in the log records linking to

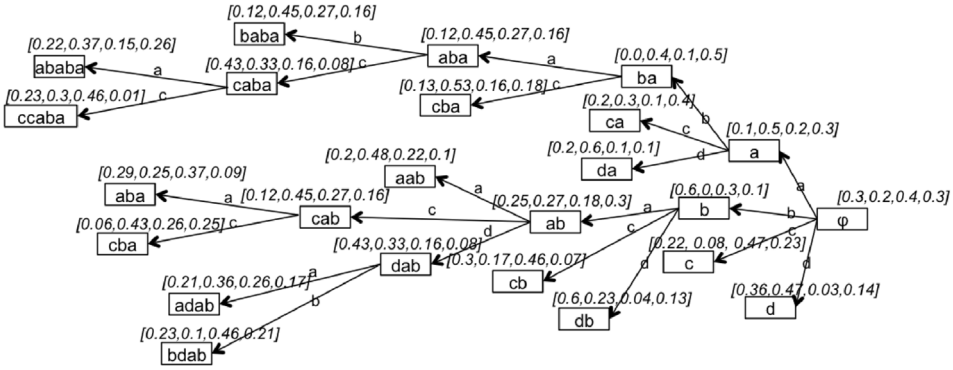

Fig. 2. A PST example.

this user. A PST is a Markov process that has been widely used to model sequence data, such as DNA, protein, and speech. It follows the similar idea as *Hidden Markov Models* (HMM) but has a better computational and space efficiency with a compact expression. We explain the idea of PST and how we use it to model a user's behavior pattern as follows.

Let $\mathcal{O}$ be a set of all operations provided by the system. We use ϕ to denote the empty sequence. A PST is a tree rooted by ϕ. Each node has an outgoing degree, i.e., the number of its children ranging from 0, if it is a leaf node, to $|\mathcal{O}|$. An edge in the tree is labeled by a single operation in $\mathcal{O}$ and there is no two edges coming from the same node labeled as the same. A node in a tree is labeled as a sequence, which is generated by walking from that node to the root. Each node is associated with a probabilistic distribution vector $\vec{\gamma}$ which records the empirical probabilistic distribution of each operation in $\mathcal{O}$ after the sequence as the label of the current node, i.e., the next operation probabilities. That is, $\vec{\gamma}^s_{\sigma_i}$ is the probability of the user performing operation σ_i after performing the sequence s, i.e., $P(\sigma_i|s)$. The sum of all the entries in each vector should be 1. It is possible that the vector may have some 0-value entry, i.e., if there is no observation of the occurrence of the corresponding sequence in the sequence database.

Fig. 2 shows a PST over the $\mathcal{O} = \{a, b, c, d\}$. As it is a suffix tree, the tree is drawn from right to left to conform to the order of operations in a sequence. The is rooted by the empty sequence. Each node in the tree, including the root node, is associated with a vector which records the probability distribution over the next operation, e.g., the probability the user performs the operation after performing the current sequence. For example, the probability distribution vector of ab is [0.25,0.27, 0.18, 0.3]. The four values should sum up to 1. This means that, after performing ab, the probability of the user performing a, $P(a|ab)$, is 0.25, the probability of the user performing b, $P(b|ab)$, is 0.27, the probability of the user performing c, $P(c|ab)$, is 0.18, and the probability of the user performing d is 0.3. It is worth to note that $P(b|ab)$ is not the same as $P(abb)$ as $P(aab) = P(a) \times P(b|a) \times P(b|ab)$.

We compute two types of empirical probabilities when building a PST. First, we compute the empirical probability of a subsequence over the given sequence set, i.e., the probability of observing the subsequence at any given position of any sequence in the set. It is computed as the ratio of the total occurrence of the subsequence to the occurrence of any subsequence with the same length. Let s be a subsequence s with the length of l. Suppose $\widehat{S}^i$ contains a set of sequences, s^1, s^2, …, s^m. We use l_i to denote each sequence's length. To compute total occurrences of s in a sequence s^i, we use a set of indicator variables ($\chi^i_j(s)$), which will be 1 if s is a subsequence of s^i, starting at s^i's j-th operation. That is:

$$\chi^i_j(s) = \begin{cases} 1 & \text{if } s = s^i[j]s^i[j+1] \dots s^i[j+l-1] \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The total occurrence of s in all sequences in $\widehat{S}^i$ is:

$$\chi^*(s) = \sum_{i=1}^{m} \sum_{j=1}^{l_i-l+1} \chi^i_j(s) \tag{2}$$

Therefore, the empirical probability of s over the sequence set can be computed as:

$$\widetilde{P}(s) = \frac{\chi^*(s)}{N_l}, N_l = \sum_{i \text{ s.t.}, l_i \geq l} (l_i - l + 1), \tag{3}$$

where N_l is the total number of subsequences in the sequence set $\widehat{S}^i$, which can overlap with others, with the same length of s.

Second, we compute the conditional empirical probability of observing an operation σ right after a subsequence s, $\widetilde{P}(\sigma \mid s)$. It is computed as the ratio of the occurrence of subsequence σs (the concatenation of s and σ), to the occurrence of subsequence s. That is,

$$\widetilde{P}(\sigma \mid s) = \frac{\chi^*(s\sigma)}{\chi^*(s)} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{l_i-l} \chi^i_j(s\sigma)}{\sum_{i=1}^{m} \sum_{j=1}^{l_i-l+1} \chi^i_j(s)} \tag{4}$$

We use a recursive approach to build a PST for each user in a DFS fashion, which is described in Algorithm 1. The algorithm takes the following input: (1) $\widehat{S}^i$: a set of sequences generated by scanning the entire event log and select the log records belonging to u_i; (2) $\mathcal{O}$: the set of all operations appearing in the log; (3) $\mathcal{L}$: the maximum depth of the tree. It usually refers to the maximum memory length or maximum length of the sequences in $\widehat{S}^{u_i}$; (4) $P_{\min}$: the threshold to identify *significant* nodes; (5) $\gamma_{\min}$: the smoothing factor. It is used to avoid having zero probability of predicting some sequence, no matter whether the sequence can be observed in the sequence set; (6) τ and ϵ: parameters working together with $\gamma_{\min}$, to remove the *useless* nodes.

The tree starts as an empty tree. A node with the label of ϕ is generated as the root. A recursive function, *traverse()* is then applied to process the current node (Lines 4–16). This process first checks whether the current node should be inserted into the tree based on three conditions. First, the node needs to be *significant*, i.e., the empirical probability of the node's label exceeds the threshold $P_{\min}$ (Line 12), to prune the PST since its size can increase exponentially with $|\mathcal{O}|$. This also allows us to cope with noises existed in the sequences, caused by accidental errors, exceptions or other factors. This condition is evaluated on every non-root node. Second, the node needs to have a child node that should be considered, i.e., the conditional empirical probability of one of its children exceeds the threshold $(1+\tau)\gamma_{\min}$ (Line 19). This is because if none of its children will be included in the tree, there is no need to maintain $\overrightarrow{\gamma}$ for the node. Therefore there is no need to include this node in the tree either, i.e., the node is *useless*. This condition is evaluated on every node. Third, the node needs to have a sufficiently higher empirical probability of predicting an operation than its parent (Line 20). As the labeling rule we mentioned before, given a node with the label $s = s[1]s[2]\ldots s[n]$, its parent node is labelled by the longest suffix of s, i.e., $s[2]s[3]\ldots s[n]$. We use *suffix*(s) to denote s's longest suffix. Otherwise, it does not make sufficient contribution to predicting a sequence to be included in the PST, i.e., the node is *useless*. This condition is evaluated on every non-root node.

Algorithm 2 Build a User Behavior Pattern PST
Input: $\widehat{S}^{u_i}$, $\mathcal{O}$, $\mathcal{L}$, $P_{\min}$, $\gamma_{\min}$, τ, ϵ
Output: a PST tree T

```
1:  T starts as an empty tree,
2:  create a new node, label it as φ {it is the root node}
3:  traverse(root)
4:  Procedure traverse(n)
5:  s = n.label;
6:  if checkValid(s) == true then
7:     add n to T {insert a node if it is significant and useful}
8:  end if
9:  for all σ ∈ O do
10:        computer P̃(σ | s) to update γ⃗^n
11:        smooth(γ⃗^n) {smooth the empirical probabilities}
12:     if P̃(σs) ≥ γ_min then
13:        create a new node, n', with the label of σs
14:        traverse (n')
15:     end if
16:  end for
17:  Function Boolean checkValid(s)
18:  for all σ ∈ O
19:     if P̃(σ | s) ≥ (1 + τ)γ_min then
20:        if P̃(σ|s) / P̃(σ|suffix(s)) > 1 + ε then
21:           return true;
22:        end if
23:     end if
24:  end for
25:  return false;
26:  Procedure smooth(γ⃗^n)
27:  c is the number of zero entries in γ⃗^n.
28:  for all γ⃗^n[i] do
29     if γ⃗^n[i] == 0 then
30:        γ⃗^n[i] = γ_min
31:     else
32:        γ⃗^n[i] = γ⃗^n[i] × (1 − c × γ_min)
33:     end if
34:  end for
```

Once a node n is inserted into the PST and its conditional empirical functionality vector, $\overrightarrow{\gamma}^n$, will be computed (Line 10). It is possible that $\overrightarrow{\gamma}^n$ has entries with zero value, if $\chi^*(s\sigma) = 0$ in Equation (4), i.e., the sequence $s\sigma$ is not observed in the sequence set. Since predicting a sequence over a sequence set is done by multiplying the related possibilities together, which we will describe later in this section, we should avoid zero probability. The vector is then smoothed by assigning $\gamma_{\min}$ to each zero entry and update others

so that $\sum_i(\overline{\gamma}^n[i]) = 1$ (Lines 11, 26–34).

After the smoothing process, all of the children of the current node will be traversed and inserted into the PST if valid (Lines 12–15). The labels of the children is generated by putting an operation from $\mathcal{O}$ to the **beginning** of the current node's label. For example, if the current node is labeled as *ab*, the labels of its children nodes are *aab*, *bab*, *cab*, and *dab*, given $\mathcal{O} = \{a, b, c, d\}$. That is, the tree is generated in *reverse* order of operations in sequences.

Complexity Analysis: To compute Equations (1) and (2), all the sequences will be checked. For each sequence, all its sub-sequences with the length of l needs to be compared with s, where the computational complexity of each comparison is O(l). Therefore, the computational complexity of computing $\widetilde{P}(s)$ is approximately $O(m * l^2)$, where m is the number of sequences in the set. Along with the same line, the computational complexity of computing $\widetilde{P}(\sigma \mid s)$ is approximately $O(m * l^2)$.

When checking the validity of a node, it needs to compute $\widetilde{P}(\sigma s)$ for all $\sigma \in O$, and possibly, $\widetilde{P}(\sigma \mid s)$ and $\widetilde{P}(\sigma \mid suffix(s))$. Therefore, the computational complexity of the function checkValid is $O(|S|* m * l^2)$. As this function costs the most in the entire PST building process, the computational complexity of the entire algorithm is: $O(|N|*|S|* m * (l\prime)^2)$, where N is the set of nodes in the generated PST and $l\prime$ is the average length of the sequences in the PST.

Therefore, the computational cost of building a PST depends on the number of nodes in the PST, the number of operations in the vocabulary, the number of sequences in the log record, and the average length of the sequences in the PST nodes.

5.2. Measure sequence similarity

Using Algorithm 2, we can build a PST tree for each user that captures how they use the system. We can then model a sequence as a PST vector, where each entry is the probability of using the corresponding PST tree to predict the sequence. That is, $v_{s_i} = [p_{i,1}, p_{i,2}, \ldots, p_{i_n}]$, where n is the number of total users and $p_{i,j}$ is the probability of using the j-th user's PST to predict the sequence s_i.

The probability of the prediction is computed as:

$$P^T(s) = \prod_{i=1}^{n} P^T(s[i]|s[1] \ldots s[i-1]), \tag{5}$$

For example, $P^T(abdcab) = P^T(a) \times P^T(b|a) \times P^T(d|ab) \times P^T(c|abd) \times P^T(a|abdc) \times P^T(b|abdcab)$. As T may not contain all the sequences, due to the pruning process, the conditional empirical probability is computed as:

$$P^T(s[i]|s[1] \ldots s[i-1]) = \overline{\gamma}^j(s[i]) \tag{6}$$

s^j is the sequence which labels the deepest node reached by walking from the root, following the path: $s[i-1]$, …, $s[1]$. For example, using the PST tree in Fig. 2, $P^T(a) = 0.3$; $P^T(b|a) = 0.5$; $P^T(d|ab) = 0.18$; $P^T(c|abd) = P^T(c|d) = 0.03$ (since there is no node labeled as 'abd' in the tree and the deepest node reached is the one labeled as 'd'); $P^T(a|abdc) = Pa|c = 0.22$; and$P^T(b|abdcab) = P^Tb|cab = 0.45$.

Sequences may have various length, we normalize the predicting probability in length using the log function. That is,

$$p_{i,j} = \frac{\log P^{T_j}(s_i)}{n_i}, \tag{7}$$

where n_i is the length of s_i. For example, the normalized predicting probability for sequence 'abdcab' using the PST tree in Fig. 2 is −0.68.

Based on our previous analysis, sequences belonging to the same workflow model should have the similar predicting probability by each PST. Therefore, we choose to use cosine similarity to compare two sequences. That is,

$$Sim_u(s_i, s_j) = \frac{v_{s_i} \cdot v_{s_j}}{\|v_{s_i}\| \times \|v_{s_j}\|} = \frac{\sum_t p_{i,t} \times p_{j,t}}{\sqrt{\sum_t (p_{i,t})^2} \times \sqrt{\sum_t (p_{i,t})^2}} \tag{8}$$

Using Equation (8), we can generate an $N \times N$ sequence similarity matrix $\mathcal{M}_u$, where N is the number of total sequences and $M_{i,j}$ is the similarity between the i-th sequence and the j-th sequence. Based on this matrix, sequences can be clustered using data clustering algorithm, such as K-Means.

6. UBP-sequence co-clustering method

In this section, we present an approach that uses co-clustering to discover sequence clusters. It leverages the interplay of UBPs and sequences in the event log, which essentially forms a dyad, which generally refers to a pair of two elements coming from two different sets [12]. We first present how to measure the distance between UBPs and sequences and then describe the co-clustering process.

We use PST to model a user behavior pattern, which follows the idea of HMM that can be used to predict the probability of sequences. Therefore, it is reasonable to measure the distance between a UBP and a sequence by checking how possible that a sequence can be predicted by the corresponding PST. That is, the higher probability of the prediction, the closer, or more relevant, the sequence and the UBP are. Therefore, we can use Equation (7) for the similarity measure. Since $P^T(s)$ is always less than 1, the

result of the log function will be always *negative*. The co-clustering algorithm we use, which we will describe in this section, requires non-negative values, we compute the absolute value of the similarity between s and T, as their *distance*. That is:

$$Dis(T,s) = |\frac{\log(P^T(s))}{n}|, \tag{9}$$

where n is the length of s.

Based on Equation (9), we generate a $M \times N$ distance matrix $\mathbf{A}$, where M is the number of users, N is the number of distinct sequences in the entire log, and $\mathbf{A}_{ij}$ is the distance of i-th user's PST and j-th sequence. Matrix $\mathbf{A}$ provides a measure on the dyad (UBP, sequence), where each entry $\mathbf{A}_{ij} \in \mathbf{A}$ indicates the relevance of the UBP T_i and sequence s_j. The most commonly known dyad may be the (document, term) pair, which is usually measured by the frequency of the term in the document or the TF-IDF score computed based on term frequency. Similarly, the measure also indicates the relevance between the document and the term. In this regard, the similarity between elements within one set can be evaluated based on their relevance to the elements in another set. For example, two documents are similar if they are relevant to a similar set of terms and two terms are similar if they are relevant to a similar set of documents. The same rationale can be applied to the (UBP, sequence) dyad.

The dyadic nature of the UBP-sequence matrix makes co-clustering an ideal tool to discover workflow models from the event logs. Co-clustering simultaneously clusters UBPs and sequences, which allows the two clustering processes to leverage each other's result to improve the overall clustering accuracy. More intuitively, since the similarity among sequences is determined by whether they are related to a similar set of UBPs, sequence clustering benefits from UBP clustering as the latter generates the sets of similar UBP that will be used for sequence similarity computation. Meanwhile, UBP clustering also benefits from sequence clustering as the same rationale applies. These two clustering processes will co-evolve and guide each other to better UBP and sequence clusters, respectively, when the co-clustering algorithm converges.

Among the existing matrix factorization techniques, including Singular Value Decomposition (SVD) [8], Latent Semantic Indexing (LSI) [15], and so on, we propose to exploit non-negative matrix factorization (NMF) [17] for UBP and sequence co-clustering as it respects the nonnegative nature of the measure on the (UBP, sequence) dyad (i.e., all entities in matrix $\mathbf{A}$ are nonnegative). We explain the idea of using NMF to co-cluster UBP and sequences as follows (see Fig. 3).

After the scaling $\mathbf{A} \rightarrow \widehat{\mathbf{A}}$, i.e., $\sum_{i,j}\widehat{\mathbf{A}}_{ij} = 1$, NMF performs 3-factor decomposition on $\widehat{\mathbf{A}}$, which factorizes it into three components:

$$\widehat{\mathbf{A}} \approx \mathbf{TBS}^T, \text{subject to} \sum_k \mathbf{T}_{ik} = 1, \sum_k \mathbf{S}_{kj} = 1. \sum_k \mathbf{B}_{kk} = 1. \tag{11}$$

where $\mathbf{T}$ is the UBP coefficient matrix, $\mathbf{B}$ is the prototype matrix, and $\mathbf{S}$ is the sequence coefficient matrix. The entries in the two coefficient matrices $\mathbf{T}$ and $\mathbf{S}$ denote the degrees of the UBPs and sequences associated with their respective clusters. The prototype matrix $\mathbf{B}$ provides a compact representation of the distance matrix $\mathbf{A}$, capturing the relevance between UBP clusters and sequence clusters.

However, simply performing a 3-factor decomposition on $\mathbf{A}$ results in a large number of equivalent factorization of the matrix, some of which may not provide proper cluster assignments [4]. For example, given a factorization $\widehat{\mathbf{A}} = \mathbf{TBS}^T$, another possible factorization is $\widehat{\mathbf{A}} = \widetilde{\mathbf{T}}\widetilde{\mathbf{B}}\widetilde{\mathbf{S}}^T$, where $\widetilde{\mathbf{T}} = \mathbf{TX}$, $\widetilde{\mathbf{B}} = \mathbf{YB}$, and $\mathbf{XY} = \mathbf{I}$. Since there are infinite number of matrices like $\mathbf{X}$, $\mathbf{Y}$, the cluster assignment from NMF can be arbitrary. An effective strategy to guarantee unique factorization results and proper cluster assignment is to enforce orthogonality on both $\mathbf{T}$ and $\mathbf{S}$. That is,

$$\mathbf{T}^T\mathbf{T} = \mathbf{I}, \qquad \mathbf{S}^T\mathbf{S} = \mathbf{I} \tag{12}$$

It has been demonstrated that applying bi-orthogonal NMF to a matrix is equivalent to perform K-means clustering on the rows and columns of the matrix simultaneously [4]. Therefore, we propose to compute the bi-orthogonal NMF of the distance matrix $\widehat{\mathbf{A}}$, which achieves the co-clustering of UPBs (i.e., rows of $\widehat{\mathbf{A}}$) and sequences (i.e., columns of $\widehat{\mathbf{A}}$). In particular, the bi-orthogonal NMF of $\mathbf{A}$ can be obtained by solving the following problem:

$$\min_{\mathbf{T},B,S\geq 0} \|\mathbf{A} - \mathbf{TBS}^T\|_F^2, \text{ s.t. } \mathbf{T}^T\mathbf{T} = \mathbf{I}, \mathbf{S}^T\mathbf{S} = \mathbf{I} \tag{13}$$

$$\begin{pmatrix} 0.3152 & 0.0117 & 0.0856 & 1E-5 & 1E-5 \\ 0.01167 & 0.2646 & 0.1167 & 1E-5 & 0.1556 \\ 0.03949 & 0.1167 & 0.2763 & 1E-5 & 0.1556 \\ 1E-5 & 1E-5 & 1E-5 & 0.1634 & 0.08561 \\ 1E-5 & 0.0233 & 0.0233 & 0.2101 & 1 \end{pmatrix}_{\mathbf{M}} \approx$$

$$\begin{pmatrix} 0.3101 & 1E-5 \\ 0.2923 & 1E-5 \\ 0.3831 & 1E-5 \\ 1E-5 & 0.0803 \\ 1E-5 & 0.7123 \end{pmatrix}_{\mathbf{T}} \begin{pmatrix} 1.1975 & 0.0382 \\ 0.0427 & 2.4179 \end{pmatrix}_{\mathbf{B}} \begin{pmatrix} 0.3412 & 1E-5 \\ 0.3216 & 1E-5 \\ 0.4303 & 1E-5 \\ 1E-5 & 0.1415 \\ 1E-5 & 0.5945 \end{pmatrix}_{\mathbf{S}}^T \tag{10}$$

Fig. 3. An example of UBP-Sequence Co-Clustering via matrix factorization.

where the matrix norm $\|\mathbf{Z}\|_F = \sqrt{\sum_{ij}\mathbf{Z}_{ij}}$. Following a similar strategy as used in Refs. [4,16], which leverages optimization theories and auxiliary functions, we derive the following update rules for **T**, **B**, and **S**:

$$\mathbf{S}_{jk} \leftarrow \mathbf{S}_{jk}\sqrt{\frac{(\widehat{\mathbf{ATB}}^T)_{jk}}{(\mathbf{SS}^T\widehat{\mathbf{A}}^T\mathbf{TB})_{jk}}} \tag{14}$$

$$\mathbf{T}_{ik} \leftarrow \mathbf{T}_{ik}\sqrt{\frac{(\widehat{\mathbf{ASB}}^T)_{ik}}{(\mathbf{TT}^T\widehat{\mathbf{ASB}}^T)_{ik}}} \tag{15}$$

$$\mathbf{B}_{ik} \leftarrow \mathbf{B}_{ik}\sqrt{\frac{(\mathbf{T}^T\widehat{\mathbf{AS}})_{ik}}{(\mathbf{T}^T\mathbf{TBS}^T\mathbf{S})_{ik}}} \tag{16}$$

The description of the co-clustering algorithm is described in Algorithm 3.

Algorithm 3 Co-Clustering UBPs and Sequences
Input: $\widehat{\mathbf{A}}, k, m$
Output: **T**, **S**, **B**
1: Initialize **T**, **S**, **B**,
2: Iterate until converge
3: Fixing **T** and **B**, update **S** using Equation (11)
4: Fixing **B** and **T**, update **S** using Equation (12)
5: Fixing **S** and **B**, update **T** using Equation (13)
6: End.

Since (13) is convex on one of the three matrices while keeping the other two fixed, an iterative process is employed that updates **T**, **B**, and **S** in turn in each iteration. The objective function is non-increasing under these update rules. It is obvious that (13) is lower bounded. Hence, the iterative process guarantees to converge.

Complexity Analysis: The time complexity of performing co-clustering is in the order of $O(MN)$. More specifically, assume that the iterative algorithm stops after t iterations. In each iteration, the time complexity of updating **T** and **B**, and **S** is in the order of $O(kMN)$. Thus, the overall complexity will be $O(tkMN)$. Since $k, t \ll M, N$, the overall complexity is $O(MN)$.

7. Experimental study

We performed a set of experiments to assess the effectiveness of the pro-posed log sequence clustering approaches. We used a synthetic data set for a thorough performance study. We also tested the approaches on a real-world data set, which repeats the pattern we saw in the result of the performance on the synthetic data set. All experiments were carried out on a Mac Pro with 2.66 GHz Quad-Core processor and 6GB DDR3 memory under Mac OS X operating system. We compare the three proposed sequence algorithms, i.e., structured-based, user-based, and co-clustering. We also implemented a clustering algorithm that only uses edit distance and compared the accuracy among the four clustering methods.

7.1. Synthetic data set

In this section, we first describe the experimental setup for the synthetic data in subsection 7.1.1. Next, we discuss our results in subsection 7.1.2.

7.1.1. Experimental setup

We simulated an event log for a Web-based marketplace system. We designed four categories of users accessing to the system, including *administrators*, *sellers*, *regular buyers*, and *window shoppers*. We designed totally 17 distinct workflow models with 19 distinct operations. A workflow model may have 4 to 9 operations and covers sequential and iteration. Users from different categories can access to different workflow models, or access to the same workflow model with different probabilistic distributions. For example, regular buyers perform the business process for placing an order with the probability of 35%, seller do that with 20% probability, while window shoppers do that with 10% probability. More specifically, there are 5 workflow models accessed by administrators, 5 workflow models accessed by sellers, 5 workflow models accessed by regular buyers and 4 workflow models accessed by window shoppers.

We generated 5 users for each user categories, resulting in 20 users in total. In order to learn how the number of sequences affects the accuracy, we designed four datasets. In the first date set, we generated 30 sequences for each user, resulted in 600 sequences in total. In the second data set, we generated 60 sequences for each user, resulted in 1200 sequences in total. In the third data set, we generated 90 sequences for each user, resulted in 1800 sequences in total. In the fourth data set, we generated 120 sequences for each user, resulted in 2400 sequences in total. In all the data sets, we generated sequences following the probabilistic distribution on the workflow models given the user's category. To simulate the deviation of workflow instances from their models in a real system, we randomly selected sequences with the number based on the noise levels, ranging from 10% to 100%, and added noises to them.

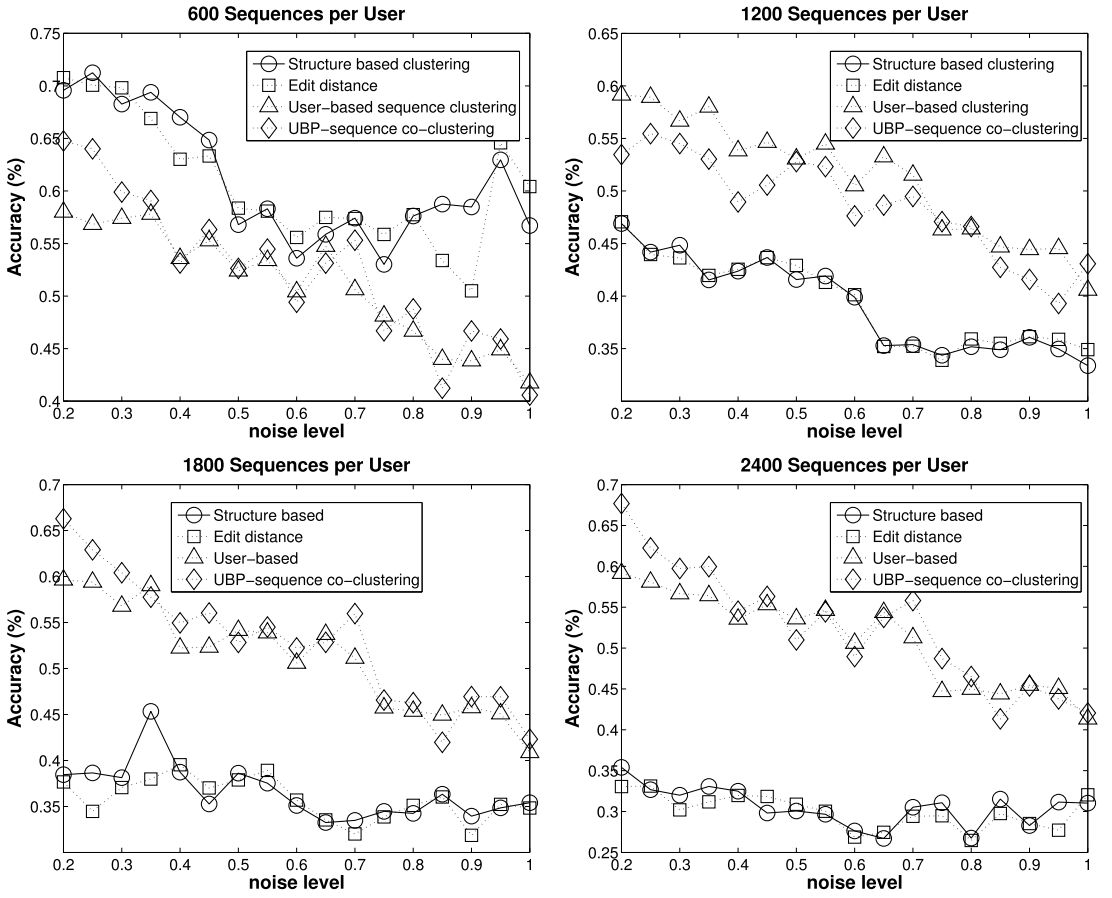

Fig. 4. *AC* performance comparison (%) vs. noise level.

We designed a noise as randomly adding operations, removing operations, reordering operations in a sequence, or performing the combination of them. For each data set with each noise level, we ran the experiment 10 times and chose the average result.

7.1.2. Results and discussion

The result was assessed by comparing resultant sequence clusters and the labels of the sequences. We adopt clustering accuracy metric for evaluation purpose, which is defined as follows:

Definition 7.1. For a given sequence $\mathbf{s}_i$, assume that its assigned sequence cluster is z_i and its workflow model is y_i. The *CA* metric is defined as follows:

$$AC = \frac{\sum_{i=1}^{N} \delta(z_i, map(y_i))}{N} \tag{17}$$

where N is the total number of sequences. $\delta(x, y)$ is the delta function that equals to one if $x = y$ and equals to zero if otherwise. $map(y_i)$ is the permutation mapping function that maps each assigned sequence cluster to the equivalent workflow model. The Kuhn-Munkres algorithm was used to find the best mapping [23].

Fig. 4 shows the comparison of the four clustering methods in terms of clustering accuracy using the four data sets. It is clearly shown that the two structured-based methods, one with edit distance and another one with adjusted edit distance, behave similarly, i.e., having the similar accuracy and the similar trend with the increase of noise levels and the number of sequences per user. This is because they are in nature the same. They both compare two sequences based on their structures, including the operations and the execution order. Our structure-based clustering method is slightly better than edit distance one since it considers the context of workflow mining and better captures the similarity. The two user-based methods, one with single-side clustering, the other with co-clustering, have the similar behavior. This is because they both model user behavior patterns and use them to determine the similarity between sequences. UBP-Sequence co-clustering performs better than single-side clustering as it involves both UBP clustering and sequence clustering, where the process of UBP clustering helps the evolvement of sequence clustering and leads to the better accuracy (see Fig. 4).

In Fig. 4, we can see that in the first data set, the structure-based method and the edit-distance method have better accuracy than user-based and co-clustering methods. The reason is that each user has only 30 sequences, which is not enough to learn a good

probabilistic model that represents the usage behavior. Therefore, the model is poor and not able to improve the clustering process. Once the number of sequences per user is more than 60, the user-based and UBP-sequence co-clustering achieves better accuracy than structured-based and edit-distance methods do. The more sequences per user has, the more significant the two former methods outperforms the two latter methods. Especially, when the log has 2400 sequences, the accuracy of UBP-sequence co-clustering method still ranges from about 0.46 to 0.68 while the edit distance method has the accuracy ranging from about 0.26 to 0.33.

Fig. 5 shows how the four clustering methods behave with the different log size. The two structure-based methods, including the ones using edit distance and the structure-based method, perform very well when the log size is small, i.e., 600 sequences in total. When the log size is doubled, the accuracy significantly dropped, e.g., from 0.7 to 0.35. The larger size the log has, the lower accuracy the methods achieve. This is because that the more sequences the log has, the more variations the log contains and the structure-based methods cannot handle those variations well. On the other hand, the two user-based methods, including the user-based sequence clustering and the UBP-sequence co-clustering, have the similar accuracies for the four data sets. Although the increase of sequence number brings more noises to the log data, it also brings more training data for learning the UBP, making it more strong and better to handle the noises. From this figure, we can see that learning UBPs and incorporating into sequence clustering process help it be robust to the noise and be more suitable for large-size log than structure-based methods.

Fig. 4 shows the results from the above three algorithms under comparison with different noise levels. We have the following important observations. First, the proposed co-clustering algorithm clearly outperforms the other two algorithms and achieves the highest clustering accuracy under all noise levels. Second, the user-based sequence clustering achieves better accuracy than the sequence only clustering, which reveals the usefulness of including user information for process mining. Third, the low accuracy achieved by the sequence only clustering algorithm confirms the challenges introduced by highly customized operation sequences in complex software systems as discussed in Section 1.

Besides the sequence clustering, the co-clustering algorithm also achieves the clustering of users. The confusion matrix as shown in Table 3 shows that the discovered user groups are perfectly in line with the four user categories. Recall that in co-clustering, sequence clustering and UBP clustering co-evolve by leveraging and benefit each other's clustering result. Therefore, sequence clustering clearly benefits from the high-quality UBP clusters obtained from UBP clustering performed at the same time and hence achieves better result than the user-based clustering.

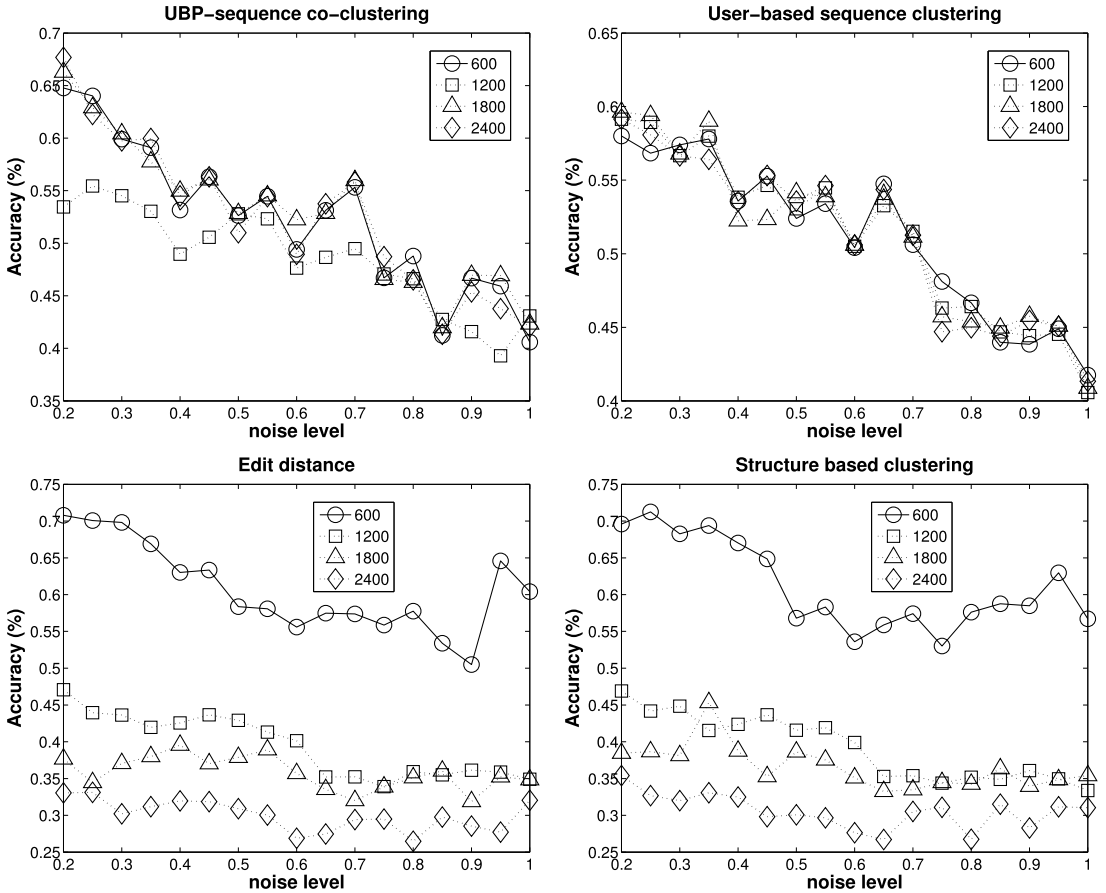

Fig. 5. *AC* performance comparison (%) vs. the log size.

Table 3
Confusion Matrix of UBP clusters.

	C_1	C_2	C_3	C_4
Administrator	5	0	0	0
Seller	0	5	0	0
Regular buyer	0	0	5	0
Window shopper	0	0	0	5

Table 4
Number of sequences per workflow.

id	workflow	Data Set 1	Data set 2
1	Manage my C.V	617	3328
2	Jobs matching my C.V	3246	13065
3	Find a job	2717	8707
4	Manage my documents	648	4431
5	Apply for a job position	1812	11735
6	Manage accepted applications	720	3806

7.2. Real-world data set

In this section, we describe the real-world data set and discuss the ground truth discovery process that we followed in subsection 7.2.1. We then follow that by a discussion of our results and findings in subsection 7.2.2.

7.2.1. Experimental setup

Although there are a number of public data sets containing event logs, most of them are not suitable for our work since either they do not include user information for various reasons or include only one or very few interaction traces for each user. Thus, they are insufficient for learning user access patterns. We chose to use the Dutch Employee Insurance Agency's data set that was published under the 2016 BPI challenge[2] as it best suits our work. The data set contains the click-data for customers of a Web-based platform that assists them in their job search on behalf of the Dutch Ministry of Social Affairs and Employment. Each entry in the log file represents a user click and contains the session id, time-stamp, user id, and the page name that the user clicked on.

Ground Truth Discovery. The data set does not provide the information that can be used as the ground truth of the experiments, including the number and the structure of workflow models, as well as the user groups. Meanwhile, as the data comes from a real-world website that serves for a large number of users, it is highly noisy. Therefore, we manually examined and cleaned the data, and then defined the ground truth.

Based on the work done in Refs. [2,13,14] and through our analysis of the click sessions, we found six representative workflow models, including *manage my CV*, i.e., the process where users visit the website mainly to fill out CV information; *jobs matching my CV*, i.e., the process where users find the jobs matching their CVs, *find a job*, i.e., the process where users can find a job based on some criteria such as job location, *manage my documents*, i.e., the process where users upload documents for job hunting or maintaining benefit status, *apply for a job position*, i.e., the process where users apply for a job that they see suitable, and *manage accepted applications*, i.e., the process where users view and manage a job position at which they were accepted. We then labeled the sequences with the workflow they belong to and filtered out those that do not belong to any workflow.

Through the analysis on the user-based access patterns, we found that users can be grouped into six clusters, including *active job seekers through matching*, *active job seekers through searching*, *indecisive job seekers with more preference on matching*, *indecisive job seekers with more preference on searching*, *active job appliers*, and *wanderers*. Users from different clusters carry out all the six workflow models described above but users within the same cluster carry them out with similar frequency. For example, active job appliers visit pages related to job application more often than active seekers. We labeled users using the groups and filtered out those that do not belong to any group.

Two Data Sets. To investigate how the proposed approaches deal with the size and the noises in the log data, we generated two data sets from the click sessions. In the first, we filtered out users with a few number of interactions since they can negatively impact the performance of the user-based clustering and UBP co-clustering approaches. Whereas, in the second dataset, we just sampled the raw data with no control over the noise level.

In both datasets, we performed the following steps for cleaning and pre-processing. First, we combined pages that have similar purpose together. For example, each posted job in the dataset has a unique link which means it's considered a separate page. However, from the perspective of user workflows, all those posted jobs are simply a user 'visiting a job post'. Thus, it is appropriate to merge those together under a single 'job-post' page. Second, we found that a large percentage of pages were rarely visited which may suggest that those pages do not represent a normal usage of the system. Thus, we took a similar approach to [2], where a similar observation was made, in which we considered only the frequently visited pages by removing those pages that exist in less than 1% of the sessions. Third, we identified and normalized the meaninglessly repetitive page clicks, e.g., a user clicked on the page "Home" around 300 times in a row. Such repetition is normalized to a singular click. Fourth, we removed those incomplete sessions, which

[2] https://www.win.tue.nl/bpi/doku.php?id=2016:challenge.

Table 5
Number of users per user-type.

id	user type	Data Set 1	Data Set 2
1	Active job seekers through matching	39	381
2	Active job seekers through searching	26	246
3	Indecisive job seekers with more preference on matching	8	391
4	Indecisive job seekers with more preference on searching	17	359
5	Active job applier	24	520
6	Wanderers	8	225

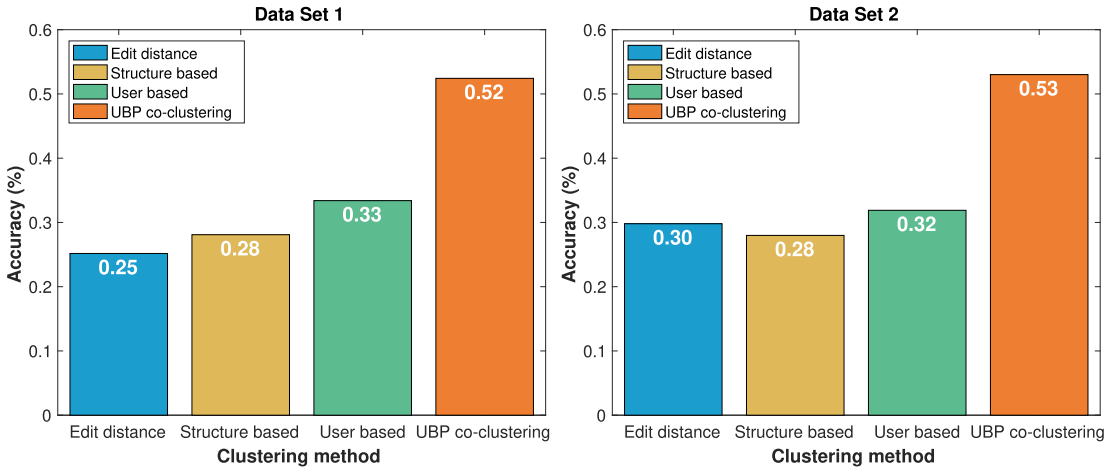

Fig. 6. *AC* performance comparison (%) of clustering method.

contain less than 3 page clicks.

In the first dataset, we have added a fifth pre-processing step in which we only selected those users and their click sessions if they have at least 80 click sessions to build their UBPs. However, in the second dataset, we just sampled the raw data and didn't put any restriction on the user selection. The number of click session for a user can be as low as 8 and averagely 22.

Tables 4 and 5 describe the two datasets.

7.2.2. Results and discussion

Fig. 6 shows the comparison of the four clustering approaches in which we ran the experiment for each method ten times and presented the average results.

First, we can see that the four approaches perform similarly in both datasets. This indicates that the approaches are scalable to the size of the data and robust to noises. Second, the structure-based clustering performs slightly better than the edit-based clustering in the first dataset and slightly worse in the second dataset. Together with the result from synthetic data, this implies that structure-based clustering performs better than edit-based clustering in a neat dataset but the advantage does not continue if the noise level is too high. Third, the performance differences between user-based and structure-based method is not as significant as in the synthetic data. We believe this is due to two reasons. First, the ground truth about the user groups are defined by our analysis on the logs, which may not be completely accurate. Second, the logs only contain the click-sessions from a single user type (i.e., the clients), where the difference of the access patterns between user groups are not very obvious. In reality, the system is used by more than the clients, such as system administrators, customer service employees, and the companies listing the jobs, like the ones in the case study. With more diverse access patterns among users, the user-based clustering approaches would be more obviously superior to structure-based ones. Nonetheless, we can see that the UBP co-clustering is clearly dominating all other clustering types which matches our observation from the synthetic data.

8. Related work

In this Section, we discuss some representative works and discuss how our work is different from them.

8.1. Process knowledge learning

In Ref. [34], an approach is proposed to discover business processes represented as Petri nets. This work uses genetic algorithms to discover some activities which cannot be derived from event logs, such as duplicate activities or hidden activities. The discovery is enabled the mapping between a Petri net and a *casual matrix*, which captures dependencies between activities, i.e., whether an

activity needs input from other activities. Once the mapping rules are defined, a genetic algorithm is used to learn the casual matrix. This work assumes that all the operations in event logs are involved in the same workflow model and they build a casual matrix to compare an operation against all other operations, which will be mapped to one Petri net. Our work assumes that an event log may contain entries for multiple workflow models where the model ids are hidden. So we cluster sequences in the log to discover multiple workflow models out of the log.

In Ref. [30], an approach is proposed to learn collaboration patterns among multiple services and identify business processes. It first uses association rule generation approach to identify frequently associated web services, i.e., the services that frequently appear in the same execution instance, to generate frequent execution flows. It then combines the execution flows and derives business process structures, such as sequential and parallel constructions. Following the Apriori algorithm, this work uses a threshold to prune the infrequent candidate service sets. This is related to the pruning idea we use when generating a PST for each user. We use a minimum empirical probability, i.e., $P_{\min}$ to remove the *insignificant* sequences. Besides the ability of finding the *significant* sequences, the PST we build defines a probabilistic distribution of sequences, which can be used to compare sequences and group them into clusters. Moreover, due to the probabilistic model we build, our approach is more robust when dealing with noises.

In Ref. [28], an approach is proposed to discover process knowledge through a probabilistic model. It focuses on learning AND-join and OR-split given the existence of hidden tasks and noise in the log. It represents the process knowledge as an AO graph, i.e., AND-join and OR-split graph and learns it through the analysis of workflow logs. The probabilistic model is used to learn if a node is a children of AND-join or OR-split, which could be invisible in the log. Algorithms were given to learn the temporal orders between nodes, such as if a node is another node's descendant, ancestor, or they are exclusive. The learning process starts with an empty tree and progressively adds nodes to the tree based on their temporal orders. Same as [34], this work assumes only one workflow model and combines all the temporal orders learned from the log into one single workflow graph. The user-based and UBP-sequence co-clustering methods described in this paper also learn temporal order through probabilistic models. However, they aims at using the temporal order to model users behavior patterns, which are used to measure sequences in order to cluster them.

8.2. Correlation discovery for workflow mining

In Ref. [27], an approach is proposed to learn the hidden correlation among data elements, which can be used to identify relevant events for workflow instances. The correlation is similar to the referential integrity in relational databases. It is derived through permutating all possible correlation candidates and evaluating each of them. The evaluation compares two attributes based on several parameters such as the edit distance of the attribute names, and the difference of the average lengths of the attribute values.

In Ref. [25], an approach is proposed to discover the correlation between information elements carried by messages, especially those that are related process execution in order to group those information elements that belongs to the same workflow instance. Given the context of web services enabled business processes, correlation relationships are learned based on the ids of workflow instances embedded in each message, or the references to the previous messages, which are supported by web service standards, such as BPEL and WS-CDL. The derived correlation relationship can be used to divide a log into workflow instances.

Both [27] and [25] aim to help identify workflow instances through discovering the link between the attributes involved in the same workflow instance. In another word, they focuses on translating data flow of workflows into the discovery of control flow. Our work puts a different focus than them. We assume that we don't have information related to the data-related aspect of a workflow, such as the input and output of each operation. We assume that each log sequence is a workflow instance, which can be directly generated from the log and tackle the problem of finding the linking between each log sequence and undiscovered workflow models.

8.3. Process sequence clustering

The approaches proposed in Refs. [10] and [6] address the similar issues as ours. They both state that there should be multiple workflow models derived from a system log, instead of just one model, and this is beyond the capability of the existing process mining approach. They both use sequence clustering algorithms to address this problem.

In Ref. [10], an approach is proposed to cluster sequences in a top down fashion. The clustering process starts with one cluster, which contains all sequences. It applies existing process mining approach to generate a process model out of them. It then measures the quality of the model to determine if it is optimal. If not, the cluster will be split into two more clusters. The approach leverages the idea of APriori algorithm, to gradually generate frequent sequences in logs, which are considered as *features*. A cluster is considered to be not optimal if a cluster contains logs having more features than the learned model. K-means algorithm is then used to split the cluster into more clusters. The sequences are compared based on their projections on features.

In Ref. [6], a sequence clustering algorithm is used to partition sequences into groups based on their similarity. The algorithm uses a first-order Markov chain to represent a cluster, which determines the membership of a sequence to the cluster. The Markov chain is initialized in a random way and get refined using Expectation-Maximization (EM) algorithm. During each iteration, for each sequence, its probability of belonging to each cluster is computed based on the Markov chain model. It is then assigned to the cluster with the highest probability. After assigning sequences to clusters, the Markov chain model of each cluster is re-calculated. The iteration will continue until all clusters converge.

Compared to these two approaches, our work is different in the way of computing the similarity of sequences and clustering these sequences. Our work incorporates user behavior patterns, UBPs, into the clustering process. Instead of only clustering sequences, we co-cluster UBPs and sequences, leveraging the similarity between UBPs to find similar sequences.

In Ref. [7], an approach is proposed to identify outlier log sequences through co-clustering patterns and sequences. Outlier

sequences do not represent normal usage of systems and need to be differentiated from those diverse ones generated from concurrent decision points. The approach first mines the frequent execution patterns from event logs, following the idea similar to association rule discovery. It then co-clusters the patterns and sequences by measuring the correlation for each pattern-sequence pair and then applying the Markov cluster algorithm. Those sequences that either do not belong to any cluster or belong to a cluster which contains too few sequences are identified as outliers. Our work is different in two aspects. First [7], focuses on identifying outliers but we focus on grouping sequences that are instances of the same workflow. Second [7], incorporates frequent patterns into co-clustering and we incorporate user related factors.

8.4. Workflow mining from interleaved logs

In Ref. [22], an approach is proposed to learn a workflow model from interleaved logs, where each log sequence is a mixture of multiple homogeneous workflow instances, i.e., the instances of the same workflow model. Such interleaved logs are common for parallel or distributed computing systems, such as Hadoop. The learning process starts with identifying temporal dependencies between each pair of operations, including indirect and direct backward dependency, as well as indirect and direct backward dependency. The dependencies are derived from probabilistic statistics. It then constructs a workflow model from the learned temporal dependencies. The model is then further refined through a verification process, where all the event traces are examined. A workflow model is derived which can interpret all event traces with minimal state transitions.

In Ref. [19], an approach is proposed to learn the hidden links between operations and hidden workflow models from interleaved log, where each log sequence is a mixture of heterogeneous workflow instances, i.e., the instances of different workflow models. Such interleaved logs are common for web applications, where users are allowed to perform multiple workflows in each session. The links are derived through a machine learning approach. Specifically, each log session is modeled as a probabilistic mixture over multiple workflows, each workflow is modeled as probabilistic distribution over operations. The posterior distributions of these models are learned through the observation of operations in each log session.

This work is different from Refs. [22] and [19] since we have different assumption and tackle different problem. Their work assume that each log sequence is a mixture of several homogenous (in Ref. [22] or heterogeneous (in Ref. [19] workflow instances and aims to decompose it into subsequences, where each subsequence is a workflow instance. We assume that each log sequence is one workflow instance and our goal is to cluster all log sequences to find all workflow instances for a hidden workflow model.

9. Conclusion

We presented three types of log sequence clustering algorithms that aim to find the links between log sequences and hidden workflow models, including sequence-based, user-based, and UBP-sequence co-clustering. As the major contribution of our work, we incorporated user-related factors into the clustering process in order to deal with noises in logs. Specifically, the structure-based clustering method compared two sequences based on the operations in them and their temporal orders. In the user-based clustering method, we modeled a user's behavior pattern as a probabilistic model over sequences, represented as a probabilistic suffix tree (PST), and built it from event logs. We then modeled each log sequence as a PST vector, where each entry is the probability of using the PST to predict the sequence. In the UBP-sequence co-clustering method, we compared a PST and a sequence based on the prediction probability and applied a co-clustering algorithm based on Non-Negative Matrix Factorization (NMF), which performs a 3-factor factorization of the matrix, to simultaneously cluster users and sequences. As indicated by our experimental study, the user-based and UBP-sequence co-clustering approaches outperformed and were more robust to noises than the structure-based method. The result showed that incorporating user-related information is effective to improve clustering accuracy. The idea of this paper can also be applied to sequence clustering in other domains.

This work focuses on the workflow models where there is only one user performing a workflow instance. we plan to extend our work to collaborative workflow models where there are multiple users involved in a workflow instance. We plan to apply machine learning approaches to learn the relation between users and operations and their roles in a business process. We will then leverage the learning result to discover the workflow models.

References

[1] G. Bejerano, G. Yona, Variations on probabilistic suffix trees: statistical modeling and prediction of protein families, Bioinformatics 17 (1) (2001) 23–43.
[2] S. Dadashnia, J. Evermann, P. Fettke, P. Hake, N. Mehdiyev, T. Niesen, Identification of distinct usage patterns and prediction of customer behavior, in: Sixth International Business Process Intelligence Challenge (BPIC'16). Business Process Intelligence Challenge (BPIC-2016), Located at BPI/BPM, 2016.
[3] I.S. Dhillon, Co-clustering documents and words using bipartite spectral graph partitioning, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '01, ACM, New York, NY, USA, 2001, pp. 269–274.
[4] C. Ding, T. Li, W. Peng, H. Park, Orthogonal nonnegative matrix t-factorizations for clustering, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, ACM, New York, NY, USA, 2006, pp. 126–135.
[5] S.R. Eddy, Hidden markov models, Curr. Opin. Struct. Biol. 6 (3) (1996) 361–365.
[6] D. Ferreira, M. Zacarias, M. Malheiros, P. Ferreira, Approaching process mining with sequence clustering: experiments and findings, in: Proceedings of the 5th International Conference on Business Process Management, BPM'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 360–374.
[7] F. Folino, G. Greco, A. Guzzo, L. Pontieri, Mining usage scenarios in business processes: outlier-aware discovery and run-time prediction, Data Knowl. Eng. 70 (12) (Dec. 2011) 1005–1029.
[8] G.H. Golub, C. Reinsch, Singular value decomposition and least squares solutions, Numer. Math. 14 (5) (Apr 1970) 403–420.
[9] G. Greco, A. Guzzo, L. Pontieri, Coclustering multiple heterogeneous domains: linear combinations and agreements, IEEE Trans. Knowl. Data Eng. 22 (12) (2010) 1649–1663.

[10] G. Greco, A. Guzzo, L. Pontieri, D. Sacca, Discovering expressive process models by clustering log traces, IEEE Trans. Knowl. Data Eng. 18 (8) (Aug. 2006) 1010–1027.
[11] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, M.-C. Shan, Business process intelligence, Comput. Ind. 53 (3) (Apr. 2004) 321–343.
[12] T. Hofmann, Learning from dyadic data, in: The Proceeding of Advances in Neural Information Processing Systems 11, MIT Press, 1998, pp. 466–472.
[13] A. Jalali, Exploring different aspects of users behaviours in the Dutch autonomous administrative authority through process cubes, in: Business Process Intelligence (BPI) Challenge, 2016.
[14] G. Janssenswillen, M. Creemers, T. Jouck, N. Martin, M. Swennen, Does Werk. Nl Work? 2016.
[15] T.K. Landauer, S.T. Dumais, A solution to plato's problem: the latent semantic analysis theory of the acquisition, induction, and representation of knowledge, Psychol. Rev. 104 (1997) 211–240.
[16] D. Lee, H.S. Seung, Algorithms for non-negative matrix factorization, in: NIPS, MIT Press, 2000, pp. 556–562.
[17] D.D. Lee1, H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 401 (1999) 788–791.
[18] C. Li, M. Reichert, A. Wombacher, Mining business process variants: challenges, scenarios, algorithms, Data Knowl. Eng. 70 (5) (May 2011) 409–434.
[19] X. Liu, Unraveling and learning workflow models from interleaved event logs, in: 2014 IEEE International Conference on Web Services, ICWS, 2014, Anchorage, AK, USA, June 27-July 2, 2014, 2014, pp. 193–200.
[20] X. Liu, C. Ding, Learning workflow models from event logs using co-clustering, Int. J. Web Serv. Res. 10 (3) (2013) 42–59.
[21] X. Liu, H. Liu, C. Ding, Incorporating user behavior patterns to discover workflow models from event logs, in: IEEE International Conference on Web Services (ICWS 2013), 2013.
[22] J.-G. Lou, Q. Fu, S. Yang, J. Li, B. Wu, Mining program workflow from interleaved traces, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 613–622.
[23] L. Lovasz, Matching Theory (North-Holland Mathematics Studies), Elsevier Science Ltd, 1986.
[24] R.S. Mans, M.H. Schonenberg, M. Song, W.M.P. van der Aalst, P.J.M. Bakker, Application of Process Mining in Healthcare – a Case Study in a Dutch Hospital, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 425–438.
[25] H.R. Motahari-Nezhad, R. Saint-Paul, F. Casati, B. Benatallah, Event correlation for process discovery from web service interaction logs, VLDB J. 20 (3) (June 2011) 417–444.
[26] D. Ron, Y. Singer, N. Tishby, The power of amnesia: learning probabilistic automata with variable memory length, in: Machine Learning, 1996, pp. 117–149.
[27] S. Rozsnyai, A. Slominski, G.T. Lakshmanan, Discovering event correlation rules for semi-structured business processes, in: Proceedings of the 5th ACM International Conference on Distributed Event-based System, DEBS '11, 2011, pp. 75–86.
[28] R. Silva, J. Zhang, J.G. Shanahan, Probabilistic workflow mining, in: ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2005, pp. 275–284.
[29] C.A. Sugar, G.M. James, Finding the number of clusters in a dataset: an information-theoretic approach, J. Am. Stat. Assoc. 98 (463) (2003) 750–763.
[30] R. Tang, Y. Zou, An approach for mining web service composition patterns from execution logs, in: Proceedings of the 12th IEEE International Symposium on Web Systems Evolution, WSE 2010, September 17-18, 2010, IEEE Computer Society, Timisoara, Romania, 2010, pp. 53–62.
[31] W. van der Aalst, T. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1128–1142.
[32] W.M. van der Aalst, Process mining: discovering and improving spaghetti and lasagna processes, in: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Paris, France, 2011.
[33] W.M.P. van der Aalst, Process mining in the large: a tutorial, in: Business Intelligence: Third European Summer School, Springer International Publishing, 2014, pp. 33–76.
[34] W.M.P. van der Aalst, A.K.A. de Medeiros, A.J.M.M. Weijters, Genetic process mining, in: 26th International Conference, ICATPN, 2005, pp. 48–69.
[35] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, W.M.P. van der Aalst, The ProM framework: a new era in process mining tool support, in: Proceedings of the 26th International Conference on Applications and Theory of Petri Nets, ICATPN'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 444–454.
[36] G.M. Veiga, D.R. Ferreira, Understanding spaghetti models with sequence clustering for prom, in: Business Process Management Workshops: BPM 2009 International Workshops, Springer Berlin Heidelberg, Ulm, Germany, 2009, pp. 92–103.
[37] M. Yan, K. Ye, Determining the number of clusters using the weighted gap statistic, Biometrics 63 (4) (2007) 1031–1037.