Process Mining with Sequence Clustering

Miguel Malheiros

Organizational Engineering Center, INESC, Portugal

Abstract. Widespread use of transactional information systems has left companies with an enormous amount of stored logging data resulting from user interaction with these systems. Although these data are kept mainly for technical purposes, there is an enormous potential in using them to discover the real business processes running within an organization. Process mining relies on data mining techniques to extract process models (or any kind of process description) from event logs such as ERP audit trails. These techniques can be used to identify business processes that have never been designed, to detect deviations from an a priori process description or to enrich a previous model with additional information. In this paper we propose an approach to process mining that, from an event log containing event records originating from different tasks or processes, tries to identify sequences of related events in a completely automatic way and apply a sequence clustering algorithm to these sequences.

1. Introduction

Current transactional information systems (e.g. Enterprise Resource Planning, Supply Chain Management, Customer Relationship Management and Workflow Management Systems) are capable of recording with great detail the interactions users have with them and store enormous amounts of records, which are called logs. The underlying implicit information that they contain and its potential use has created the necessity to develop tools which are able to mine this information and make it explicit with the help of some formal representation. Process mining is one such tool. This technique takes event logs from a business process as its input and tries to generate a model for the business process that generated the logs. A business process is a set of coordinated tasks and activities, conducted by both people and equipment, that will lead to accomplishing a specific organizational goal (SearchCIO 2007).

The generated process models can be used to formalize a process which had never been designed, to make a delta analysis of the process (i.e. evaluate the differences between the designed process and the real process) or to improve an already known model. Contrary to the traditional approach of first designing the process and then applying it process mining tries to first capture the real process from real execution and then formalize it (Figure 1).

However, there are some obstacles to generating the process models. Some processes are intrinsically complex and therefore are very difficult to mine. Also, incorrect logs (noise) can cause the mined model to contain errors.
or be inaccurate. Moreover, some processes contain activities which are not executed in computers and cannot be automatically logged, however we will abstract from this issue.

Most authors’ approaches assume that an event log only contains events from a single business process, i.e., that each event has an associated case or process instance. This comes as a major disadvantage since (1) the classes of information systems that are able to generate such logs are restricted to process-aware systems, and (2) it becomes impossible to apply and benefit from process mining in scenarios where the log data is not available in that form (Ferreira 2007). In this paper we present an approach that makes no such assumption.

In this paper we propose an approach to process mining that, from an event log containing event records originating from different tasks or processes, tries to identify sequences of related events in a completely automatic way and apply a sequence clustering algorithm to these sequences. The sequence clustering algorithm we used groups similar sequences in clusters and constructs a model, in the form of a Markov chain, for each cluster. No input information about the business logic of these events is provided or necessary. As a result, the models found do not constitute perfect models of the behavior that generated the events, but illustrate approximately what types of sequences were executed and their meaning. The sequence clustering algorithm is discussed in section 2.

In order to generate event logs, we developed an application that executed sequences of actions over a small database and recorded these actions in logs. By looking at these logs alone it is impossible to know exactly what sequences originated them since they are just a long list of events. However, as we know the types of sequences that originated the logs, we have a way of assessing the quality of our approach by comparing the models found to the original sequences we programmed, and therefore we were able to make adjustments accordingly. The simulator application is described in section 3.

After capturing the event log it is necessary to determine what sequences are in it. In section 4 we present our approach to this problem. We analyze all events and establish connections between them according to a specific criterion. The resulting graphs, where states are events and edges are event connections constitute sequences of actions when its nodes are sorted chronologically.

In section 5 we discuss and show the results obtained. General conclusions and future work are presented in section 6.

2. Sequence Clustering

Sequence clustering is a data mining technique that takes a number of sequences and groups them in clusters so that each clusters contains similar sequences. A sequence is a series of discrete states (Tang 2005), one example is a genomic sequence where the states are adenosine, guanine, cytosine and thymidine. The applicability of this technique to genomic sequences explains its widespread use in the bioinformatics field. Sequence clustering is also a probabilistic technique, since that in the sequence models produced the transitions between states have associated probabilities. This makes these models robust to noise, since a transition with a low probability indicates a sequence path that is rarely followed, thus probably an erroneous one. Furthermore, the fact that this algorithm takes sequences as input seems to make it a natural candidate to mine sequences of actions, which in turn can be obtained from event logs.

Microsoft SQL Server 2005 includes a tool for sequence clustering that makes use of the Microsoft Sequence Clustering algorithm and which we decided to use in our approach. In Microsoft Sequence Clustering algorithm each case is assigned to each cluster with some probability. Each cluster has an associated Markov chain. A Markov chain is described as a set of states and the transition probabilities between them. If the chain is currently in state $s_i$, then it moves to state $s_j$ at the next step with a probability denoted by $p_{ij}$, and this probability does not depend upon which states the chain was in before the current state (Grinstead 1997). Figure 2 shows a Markov chain.
The definition presented above is actually the definition of a first order Markov chain. In a $n^{th}$ order chain the future state’s probability depends on the previous $n$ states.

The Microsoft Sequence Clustering algorithm involves the following steps (Tang 2005):

1. Initialize the model parameters at random, i.e. initialize the state transition probabilities of the Markov chains from each cluster at random;
2. Assign each case to each cluster with some probability;
3. Recalculate the model parameters taking into consideration the probabilities of each case belonging to each cluster, i.e. recalculate the state transition probabilities of each Markov chain of each cluster considering the probabilities of each case belonging to that cluster.
4. Verify if the model has converged, in case it has not, reiterate from step 2.

In order to work the algorithm needs at least two tables, a table with the non-sequence attributes (the case table) and a nested table of the first with the sequence key. There can be only one sequence nested table in each model. An example is shown in Figure 3, the goal is to model the behavior of a TV viewer when he is zapping between channels. This is similar to the example presented in (Tang 2005). The Viewer table contains general data – non-sequence attributes - about the viewer. Table ZappingSequence is the sequence nested table and it contains three attributes, ViewerID which is the foreign key to the main table; ChannelType which is the state of the sequence; and SequenceID which saves the sequence position of a state.

For example, if viewer number 1 zapped between channels CNN, Eurosport, MTV, BBC and SporTV in this order and viewer 2 zapped between SkyNews, VH1 and ESPN in this order the tables would look like this:
The algorithm is tunable by the following parameters: number of clusters the model contains; minimum number of cases in each cluster; maximum number of different states an attribute can have; and the maximum number of different states the sequence attribute can have.

The process mining activity begins with an event log. Unfortunately, we cannot apply the sequence clustering algorithm directly to the log. As we already explained, we need to have a case table and a nested table. This way, several steps must be taken in order to preprocess the log and be able to build these two tables from it. The next sections explain how we generated an event log using a simulator application, all the steps that had to be taken to construct the case and nested tables and the result obtained after finally applying the sequence clustering technique.

3. Data Collection

The first step of our approach involved the development of a banking activity scenario, consisting of a fictitious database and a small application that interacted with it, which could produce specific event sequences, such as: creation of a checking account, creation of a savings account, creation of a loan and payment of a loan.

The database consists of the following tables: accounts, which can be savings accounts or checking accounts and belong to a specific branch of the bank; loans (also associated to branches) and loan payments; customers, associated to loans through the borrower relationship and to accounts through the depositor relationship; and employees associated to customer through the customer-banker relationship (Figure 4).
The event log is generated by simulating several banking operations. This simulation is accomplished by executing SQL queries over the banking database just presented. The following banking operations may be executed:

1. Checking account creation
2. Savings account creation
3. Loan creation
4. Loan payment

Creating a checking account for a new customer, for example, involves the following steps: (1) create a new customer, (2) create a new account at the branch, (3) save the account as a checking account with a certain withdrawal limit, (4) associate the customer as a depositor of the account, and (5) associate an employee as account manager for that customer. In terms of SQL, this operation would look like (Ferreira 2007):

```sql
INSERT INTO Customer VALUES (85045,'John Hayes','North Street','Southampton')
INSERT INTO Account VALUES (34220,705,'Downtown')
INSERT INTO Checking_Account VALUES (34220, 207)
INSERT INTO Depositor VALUES (85045,34220)
INSERT INTO Cust_Banker VALUES (85045,6,'account manager')
```

Some sequences may be executed in more than one way. In this case, for example, steps 4 and 5 could be swapped. The checking account creation sequence has 4 versions in total. Also, it is assumed that the database is already populated when the sequences are executed.

Figure 5 illustrates the SQL Profiler event log resulting from a savings account creation sequence.

```
<table>
<thead>
<tr>
<th>EventClass</th>
<th>SQLID</th>
<th>StartTime</th>
<th>StopTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace start</td>
<td></td>
<td>2007-06-22 18:17:04...</td>
<td></td>
</tr>
<tr>
<td>sql:batchcompleted</td>
<td>82</td>
<td>SELECT branch_name, branch_city, etc...</td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
</tr>
<tr>
<td>sql:batchcompleted</td>
<td>82</td>
<td>SELECT customer_id, customer_name, etc...</td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
</tr>
<tr>
<td>sql:batchcompleted</td>
<td>82</td>
<td>SELECT account_number, balance, etc...</td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
</tr>
<tr>
<td>sql:batchcompleted</td>
<td>82</td>
<td>INSERT INTO ACCOUNT VALUES (34220)</td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
</tr>
<tr>
<td>sql:batchcompleted</td>
<td>82</td>
<td>INSERT INTO savings_account VALUES (34220)</td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
</tr>
<tr>
<td>sql:batchcompleted</td>
<td>82</td>
<td>exec internal account transfer</td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
</tr>
<tr>
<td>sql:batchcompleted</td>
<td>82</td>
<td>INSERT INTO Depositor VALUES (34220)</td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
</tr>
<tr>
<td>Trace stop</td>
<td></td>
<td>2007-06-22 18:17:08... 2007-06-22 18:17:08...</td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 5 – Profiler event log**

After the log is captured we save it into a table of our database in order to be able to manipulate it. As stated in
Section 2, we cannot apply the sequence clustering algorithm directly to this event log. By looking at the log we can only see a list of queries without any type of associated case id. It is impossible to know exactly how many sequences exist, where each one starts, ends or what events are included in it. Remember that the simulation phase is now over, and we assume that this log was generated by a real system of which we do not know anything about. The problem then, is to find a way of determining what sequences of events exist in it. After finding these sequences we must structure them in a way understandable by the sequence clustering algorithm. The next section shows how to find the hidden sequences.

4. Log Preprocessing

After capturing the log it is necessary to process it and restructure its data so that we are able to apply the Sequence Clustering algorithm. Since the log is just a long list of queries and stored procedures we were required to find a way to determine where each sequence begins and ends and which events belong to each one. An application called Sequence Builder was developed with this purpose.

After saving the log to a database table called Trace, which mimics the structure of Figure 5, the first step in log processing is to take each row and parse the TextData field in order to know what type of query it represents and what tables and attributes were manipulated by it. When this is known the program decides whether to store the query or ignore it. The program ignores a query if it is not a select, insert, delete, update or execute stored procedure command or if it is too generic to be of use, like a select, delete or update command without a where clause. For example, the query `SELECT * FROM Customer` would be ignored.

In order to support the preprocessing steps that will be described ahead, two new database tables were created to store relevant queries: Query and Keys. Figure 6 shows these two tables.

![Figure 6 - Tables Query and Keys](image)

The query `INSERT INTO Customer VALUES (85045,'John','Happy Street','Sunny City')`, for example, originates the following records:

![Figure 7 - Query table for INSERT Customer example](image)

![Figure 8 - Keys table for INSERT Customer example](image)
Note that the program only stores the value for the customer_id field, which is the primary key for table Customer. There is no need to store anything else, because the tag and the key_name, key_value pair unequivocally identify that this query is an insertion of customer 85045 in the database. The program knows how many primary keys each table has and their names because it queries the database catalog before the log processing begins and stores these data.

But how do we know two queries are connected and consequently belong to the same sequence? We simply compare each query with every other query and if they share a key_name, key_value pair we assume they’re connected.

The connections that are found are saved in table Follows (Figure 9).

<table>
<thead>
<tr>
<th>Follows</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
</tr>
<tr>
<td>PK</td>
</tr>
</tbody>
</table>

Figure 9 – Table Follows

For example, if we determine that query 1 is connected to query 2, meaning they belong to the same sequence, the table would look like this:

<table>
<thead>
<tr>
<th>query_id</th>
<th>follows</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 10 - Query 2 follows query 1 example

Since the query_id attribute indicates the order in which queries were executed, we know query 2 happened after query 1, therefore, when we determine queries 1 and 2 are connected we also know query 2 follows query 1 and not the other way around. After all possible query pairs are compared the Follows table will actually contain a graph representation of the whole log, where states are events and edges are connections between events. Figure 11 illustrates this idea for the “Open Checking Account” and “Open Savings Account” sequences.

Figure 11 - Graph view of two banking sequences. Links have the names of the parameters whose values are the same in the queries they connect. (Ferreira 2007)

It is easy to see that if a customer creates two accounts, for example, there will be connections not only among events intuitively belonging to a sequence but also between different sequences. This case is illustrated in Figure 12, where there should be three different sequences but there are only two. In order to solve this problem we developed a heuristic where very long connections are removed from table. To do this, the average connection
length is calculated and all connections that exceed a certain average based threshold are removed.

**Figure 12 - Links between events that belong to different sequences. (Ferreira 2007)**

Having removed all the long links from the Follows table the application now has to find out what specific sequences exist, i.e., the application must go through all implicit sub-graphs in this table, sort the queries by chronological order (sorting by query_id has the same effect) and store the sequence found. In order to do this the application executes the following cycle:

1. Get the first query from the Query table. If table is empty terminate;
2. Find the nodes belonging to this query’s sub-graph by using a recursive SQL query over table Follows;
3. This group of nodes constitutes a sequence and this sequence is now inserted on a case and a nested table (Figure 13).
4. Remove all queries belonging to that sequence from the Query table to avoid repeatedly finding the same sequence.

Figure 13 shows the diagram of the two tables that store the sequences that were found. Table Sequence stores the sequence_ids that exist while table Action stores the events (queries) that belong to each sequence. The position field is simply the query_id of the query represented and the action_tag is its tag.

**Figure 13 - Tables Sequence and Action.**

We have now found all the sequences present in the log and have the data structured in a case table nested table format, where the Sequence table is the case table and the Action table is the nested table.

**5. Results**

We will now present the results we obtained in applying the techniques from chapter 5 and chapter 3 over a simulator generated log of about one thousand events consisting of more or less 200 sequences. When the sequence clustering algorithm is run without specifying a number of clusters it uses a heuristic to determine the most “adequate” number of clusters. Doing this over our data resulted in the algorithm grouping nine different
sequence models in eight clusters, i.e., one of the clusters contained two Markov chains for two different types of sequences. Seeing this, we instructed the algorithm to create 9 clusters and ran it again. Each cluster had now just one model (Figure 14 and Figure 15 show 5 of the 9 clusters created).

All different types of sequences were identified correctly except one (Figure 15).

Figure 14 - Sequence models for clusters 1, 2, 3 and 4

Figure 15 - Cluster 5’s sequence model

Figure 16 shows the cluster diagram for this example. The links indicate cluster similarity. The algorithm considered clusters 2, 3, 4 and 7 all very similar. These clusters correspond to the open checking account sequence variations. As these variations only change the order of the actions and as each cluster only contains sequences corresponding to one of these variations it is easy to understand why the algorithm found them to be similar. It also found similarities between clusters 9 and 5 and between clusters 1 and 8 which correspond to the loan payment and saving account opening sequences respectively. Cluster 6 is isolated indicating the algorithm
found it to be different from all other. This cluster contains the loan creation sequences, which do not have any variations and whose events are different from the other sequences, thus justifying its isolated position.

![Cluster diagram](image)

**Figure 16 - Cluster diagram**

6. Conclusion

All the different types of sequences were identified correctly except one whose model slightly different from what it was supposed to be. Moreover, the algorithm was able to correctly identify which types of sequences were similar to others, thus indicating what types were variations of other types.

The experiments described in this report confirm that sequence clustering algorithms can be valuable tools in the sequential analysis of event logs and, consequently, in the discovery of organization’s tasks, overcoming (with the help of appropriate preprocessing techniques) certain limitations of current algorithms. In order to gain knowledge at the process level some more work will have to be done so that we can understand what additional data is necessary to connect tasks. There is also the possibility that human validation will be necessary.

All in all, the execution of the algorithm is the simplest step of the whole approach. The biggest challenge is actually to identify the set of sequences of actions in a completely automatic way. Most process mining approaches assume logs only contain events resulting from executions of a single process (task). Our approach makes no such assumption since most enterprise systems’ logs won’t contain events pertaining to just one process and also because these logs will contain events whose processes and process instances are unknown. This way, we had to develop a way of determining the case id for each event. The technique designed to that effect, matching key_name, key_value pairs in order to build graphs corresponding to sequences is the most important step of the described approach since it contributes for the mining of event logs resulting from multiple, interleaved process events.

References


