Process Mining the Credit Suisse Advisory Process

A Major Qualifying Project Report:

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By:
Shannon Feeley
Ian Jacoway
Gina Rios

Advisors:
Randy Paffenroth
Stephan Sturm

Sponsor:
Souleymane Bah, Credit Suisse

Date: April 27, 2017

Contents

Abstract

Our Major Qualifying Project is to show the business benefits of process mining, and our sponsor is Credit Suisse. Credit Suisse uses many different processes for its employees to onboard, advise, and manage its clients. Each activity that an employee completes in the process is logged in the form of an event. An event consists of a timestamp, a case ID, and an activity. A collection of events that are part of the same process is called an event log. Event logs can be analyzed, or "mined," to discover a process model. The derivation of a process model is called "process mining." Through the use of process mining, a company can compare the process model to the intended business model to identify deviations. We consult with the Credit Suisse team in charge of the advisory process to compare the process model derived from the advisory process data to their intended business model of the advisory process. The advisory process is Credit Suisse's five step process to assist their clients in creating investment portfolios. We are able to identify deviations from the intended model and suggest methods of reducing operational costs by eliminating redundant tasks and points of congestion to increase productivity.

1 Introduction

Credit Suisse is a leading wealth manager and bank. The bank focuses on several audiences, such as shareholders and clients. Credit Suisse uses many different processes for its employees to onboard, advise, and manage its clients [1].

Each activity that an employee completes in a process is logged in the form of an event. An event consists of a timestamp, a case ID, and an activity. A case is the collection of all the events that correspond to the same case ID. A sequence of activities forms a process. An example of a process is applying to college: there is an ordered set of activities that high school students must perform when they apply to college, such as writing personal statements, taking the SATs, and submitting SAT scores. Joe is applying to college, so he is a case in the process. An example event is *5:00 p.m., Joe, Submitting SAT Scores*. The timestamp is 5:00 p.m.; the case ID is Joe; and the activity is Submitting SAT Scores. A collection of events from multiple cases that are part of the same process is called an event log. Event logs can be analyzed, or "mined," to discover a process model. A process model of the college application process shows the ordered steps students take in their application process, such as sending SAT Scores, submitting their personal statement, and requesting their letters of recommendation. Creating a process model is called "process mining." Through the use of process mining, a company can compare the process model to the intended business model to identify deviations from the intended model. Process mining also allows businesses to reduce operational costs by identifying and eliminating redundant tasks and points of congestion to increase productivity. For example, Credit Suisse can compare the process model derived from the advisory process data to their intended business model of the advisory process [2].

Process mining is an area of data science that utilizes computer science and mathematics to analyze processes based on data contained in event logs [2]. Our goal is to show that process mining can be used to reduce Credit Suisse's operational costs. We use Credit Suisse's advisory process (the five-step process that employees utilize to assist clients in developing investment portfolios) to demonstrate the business value of process mining. The logs specific to the advisory process are gathered and analyzed. Through the use of a process mining tool, we derive a process model based upon a specified algorithm (the inductive miner).

The resulting process model is then used to make recommendations based on deviations from the intended business model and points of congestion found in the analysis, which demonstrates the business value of process mining.

2 Background

The following sections present the building blocks of process mining. We begin with the definition of an event log and then define process mining and its benefits. We also explain Credit Suisse's Advisory Process. This information provides the reader with necessary information to understand our project

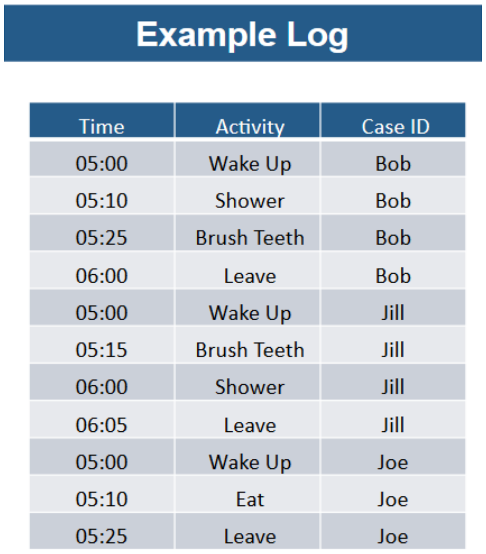
2.1 Event Log

A timestamp, a case ID, and an activity form an event [2]. For example, one event could be *05:00, Bob, Wake Up*. The timestamp is 05:00; the case ID is Bob; and the activity is Wake Up. A case is the collection of all the events that correspond to the same case ID. All of the events that correspond to Bob form a case. A collection of events from multiple cases that are part of the same process are called an event log. See figure 1.

2.2 Process Model

A sequence of activities form a process [2]. A person's morning routine, as seen in the event log example in Figure 1, is an example of a process. The morning routine process for Bob is "Wake Up," "Shower," "Brush Teeth," then "Leave." We discover a process model by analyzing the event logs of these activities. A process model combines all of the cases in an event log to represent the mainstream behavior of the event log as a single process.

Continuing our example from Figure 1, we construct a process model based on the activities of each individual as they get ready in the morning. The process model begins with every individual starting with the "Wake Up" activity.



Example Log

Time	Activity	Case ID
05:00	Wake Up	Bob
05:10	Shower	Bob
05:25	Brush Teeth	Bob
06:00	Leave	Bob
05:00	Wake Up	Jill
05:15	Brush Teeth	Jill
06:00	Shower	Jill
06:05	Leave	Jill
05:00	Wake Up	Joe
05:10	Eat	Joe
05:25	Leave	Joe

Figure 1: An example of an event log of three people getting ready in the morning including a time, an activity, and a case ID. Each person's morning routine is shown in the table.

Next, the individual has a choice; they can choose to "Eat" or they can choose to "Shower" and "Brush Teeth". If the individual chooses to "Shower" and "Brush Teeth", then both activities must be completed (in any order) before they can move on to "Leave". Finally, the individual exits the process model with the "Leave" activity, as depicted in Figure 2.

2.3 Advisory Process

The Advisory Process is a protocol that Credit Suisse employees utilize when assisting clients in developing their investment portfolios. There are five steps to Credit Suisse's advisory process. The first step is called "Needs Analysis", which is the assessment of a client's personal and financial situation. The second step is called "Financial Concept", which is the creation of an overview of the client's personal assets. The third step is called "Client Profile", which is the discussion of the client's willingness and ability to take on risk. The fourth step is called "Strategy", which is the presentation of potential financial strategies to

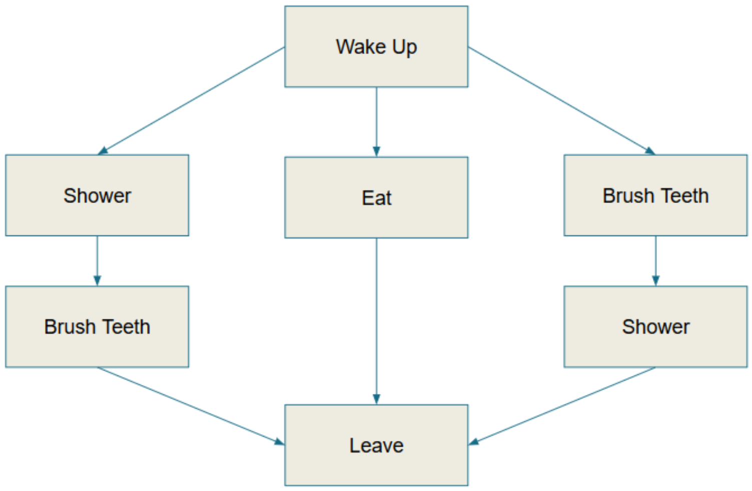
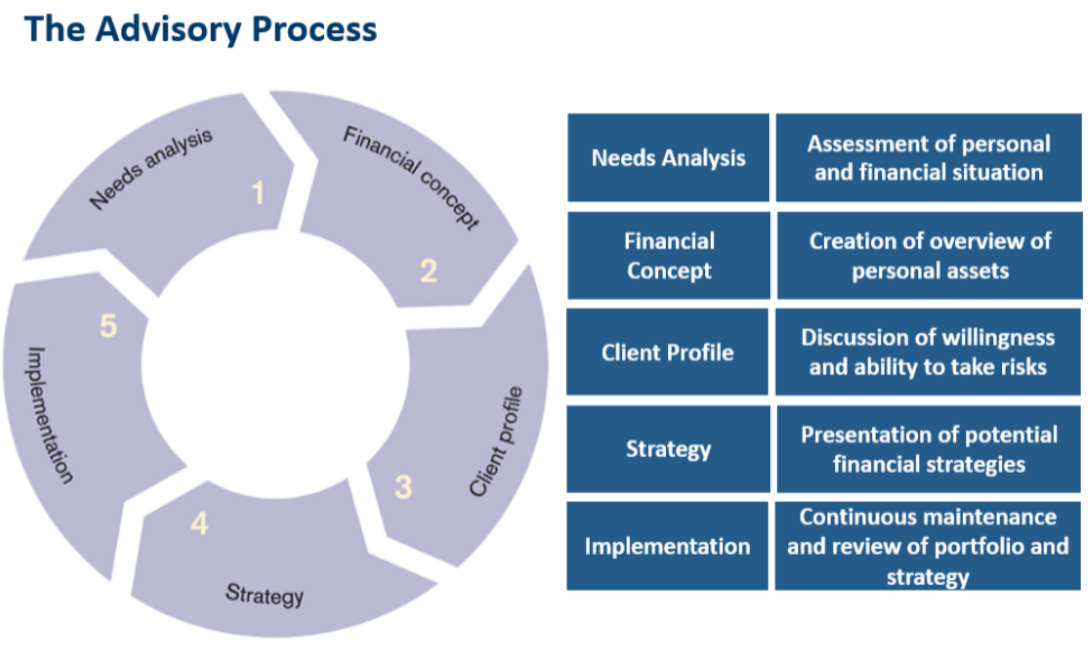


Figure 2: An example of a process model mined from the example event log in Figure 1. In the model, we see all cases start with the activity "Wake Up". Then, the cases move to the first split in the process, where they must choose to "Eat" or to "Shower" and "Brush Teeth". If the case chooses to "Shower" and "Brush Teeth", then both activities must be completed before the case can move on to "Leave" and complete the process. Finally, the case completes the process with the "Leave" activity.

the client. The final step is called "Implementation", which is the continuous maintenance and review of the client's portfolio and strategy [3]. See Figure 3 for an illustration of the advisory process.

When Credit Suisse employees execute any activity on their laptops, the activity name, the timestamp, and the user identification is logged in the Credit Suisse system. The logs specific to the advisory process are what we used for our case study.



Needs Analysis	Assessment of personal and financial situation
Financial Concept	Creation of overview of personal assets
Client Profile	Discussion of willingness and ability to take risks
Strategy	Presentation of potential financial strategies
Implementation	Continuous maintenance and review of portfolio and strategy

Figure 3: An illustration of the advisory process and definitions of the process' five steps.

2.4 Process Mining

Process mining utilizes computer science and mathematics in order to analyze processes based on data contained in event logs. The goal of process mining is to answer questions about operational processes. For example, what happened in the past and why, what will happen in the future, how can a company better control its processes, and how can a process be redesigned to improve performance [2]. Process mining uses three types of techniques to create and analyze process models from event logs. The three types of techniques are discovery, conformance checking, and enhancement. A discovery technique uses the event logs to create a model without any additional information. In other words, it represents the process strictly from the data. A conformance checking technique compares a process model to what actually occurs in the event logs. This technique determines whether or not the process model derived from the event logs matches the anticipated business model. Conformance checking determines where any deviations exist between the anticipated business model and the process model derived from the event logs. The goal of enhancement is to improve a process. There are two types of enhancement: repair and extension. Repair changes the model to better represent what occurs in reality. Extension adds perspective to the process model. An example of extension is including timestamps in log data, which extends the model to show frequencies and potential bottlenecks. We mostly focus on conformance checking to see if the data matches the intended business model [2] [4].

Process mining is usually done by computer programs, such as ProM. ProM is a free, open-source process mining tool. It allows users to import event logs and view statistics about the event log, such as the frequency of activities. Then, the user can run various algorithms on the event log to produce process models, such as the inductive miner which is described in the following section [5].

It is important for organizations to allocate resources efficiently, and process mining can be useful in finding inefficiencies. Process mining identifies bottlenecks and redundant tasks, which companies can eliminate in order to reduce operational costs and increase productivity. In addition, process mining documents business processes, which are used in conformance checking. Conformance checking enables businesses to identify deviations from their intended business model [2].

2.5 Inductive Miner

We use the inductive miner implemented in ProM to analyze the advisory process event logs. We choose to use the inductive miner because the algorithm is sound, meaning the algorithm works well with real life data and the results have provable correctness. The algorithm only uses the data to create the model, so it is not influenced by any outside information or biases. The inductive miner algorithm is also able to accurately represent the mainstream behavior of real data, which makes it easy to identify deviations from the general population's behavior. The following sections explain the inductive miner algorithm.

2.5.1 Introduction

The Inductive Miner is an algorithm in ProM that aims to represent mainstream behavior of a process. The Inductive Miner algorithm recursively splits the event log into sublogs until each activity is isolated. It first creates a directly-follows graph according to the procedure in Section 2.5.2. The directly follows graph is a directed graph, where each vertex corresponds to an activity and each directed edge/arc implies that the activity of the vertex where the edge begins occurs before the activity of the vertex where the edge ends. For example, if there is a directed edge from activity A to activity B, then activity A occurs before activity B.

After the directly-follows graph is created, the algorithm makes cuts to the graph. A cut partitions the set of vertices in a graph into distinct subsets. There are four different types of cuts, each corresponding to one of the four operators that can be present in a process tree: $\rightarrow$, $\times$, $\wedge$, and $\circlearrowleft$ (see Section 2.5.3 for more on process trees). The process operators explain the relationship between activities. For example, if one event always precedes another they are connected by the $\rightarrow$ operator. Therefore, the type of cut explains the relationship between activities. To give another example, if the algorithm splits activities using the $\wedge$ cut, then those activities are connected by the $\wedge$ operator.

For each cut, the algorithm looks for exclusive choice cuts ($\times$), then sequence cuts ($\rightarrow$), then parallel choice cuts ($\wedge$), and finally redo loop cuts ($\circlearrowleft$). It continues to make cuts until each activity is isolated. Once each activity is isolated, we know how the activities are related by the operators used to cut the graph. We then can create a process tree, which is described in detail in the following sections [2].

2.5.2 Directly-Follows Graphs

Directly-follows graphs delineate start activities, end activities, and activities which directly follow each other. Each vertex in the directly-follows graph corresponds to an activity in the event log. Each directed edge/arc indicates that the activity of the vertex the edge starts at occurs before the activity of the vertex that the edge ends at.

An event log, L, is a multi-set over the set of finite sequences over $\mathscr{A}$, L $\in \mathscr{B}(\mathscr{A}^*)$ where $\mathscr{A}$ is the set of all activities, $\mathscr{A}^*$ is the set of all finite sequences over A, a multi-set is a set that allows for multiple occurrences of the same element, and $\mathscr{B}(\mathscr{A}^*)$ is the set of all multi-sets over $\mathscr{A}^*$. For an event log L:

- $A_L = \{a \in \sigma,\ \sigma \in L\}$ is the set of activities in L.
- $\mapsto_L = \{(a,b) \in A_L \times A_L \ \mid\ a >_L b\}$ is the set of activities which directly follow each other. This means that there exists a case in the event log L where "a" is executed before "b". $>_L$ is a transitive relation between "a" and "b" that signifies "a" occurs directly before "b". We write $a \mapsto_L^+ b$ if there is a path from a to b.
- $first(\sigma)$ denotes the set of start activities. A start activity is any activity that occurs first in a case.
- $A_L^{start} = \{a \in A_L \mid \exists_{\sigma \in L}\, a \in first(\sigma)\}$ is the set of start activities. This means that there exists a case in the event log L where "a" is the first activity in the case.
- $last(\sigma)$ denotes the set of end activities, which are any activities that occur last in a case.
- $A_L^{end} = \{a \in A_L \mid \exists_{\sigma \in L}\, a \in last(\sigma)\}$ is the set of end activities. This means there exists a case in the event log L where "a" is the last activity in the case.

The directly-follows graph of L is the quadruple $G(L) = (A_L, \mapsto_L, A_L^{start}, A_L^{end})$. For example, suppose $L = ((A,B,C,D),(A,C,B,D),(A,E,D))$.

- $A_L = \{A,B,C,D,E\}$
- $\mapsto_L = \{(A,B),(A,C),(A,E),(B,C),(B,D),(C,D),(C,B),(E,D)\}$
- $A_L^{start} = \{A\}$
- $A_L^{end} = \{D\}$

We know that A is our only start activity, so our graph begins with A. Next, we look at $\mapsto_L$ to see which activities follow A. Activities B, C, and E follow A, so we construct the graph in step 1 of Figure 4. Then, we see which activities follow activity B by looking at $\mapsto_L$. Activity C and activity D follow activity B, so we update our graph shown in step 2 of Figure 4. We continue to repeat the process until all of the pairs in the $\mapsto_L$ are represented on the graph. See step 3 in Figure 4 for the completed directly follows graph [2].

2.5.3 Process Trees

Process trees are a way to represent or encode a process. Process trees make it easier to decide where loops or processes occur. A tree is a connected graph with no cycles. There are four operators that can be used in process trees; they are $\rightarrow$, $\times$, $\wedge$, and $\circlearrowleft$. Process trees are read from left to right. The entire leftmost branch is read, beginning with its leftmost child until the entire branch has been read. Then, the branch to its right is read in the same manner until the last branch. The sequential composition operator ($\rightarrow$) implies the activities are executed, from left to right (see Figure 5).

The exclusive choice operator ($\times$) implies a choice between activities where only one is executed (see Figure 6). The parallel composition operator ($\wedge$) implies that the activities can be executed at the same time or directly following each other and that all activities must be completed before moving to the next phase of the diagram (see Figure 7). If activity A, activity B, and activity C are related by the

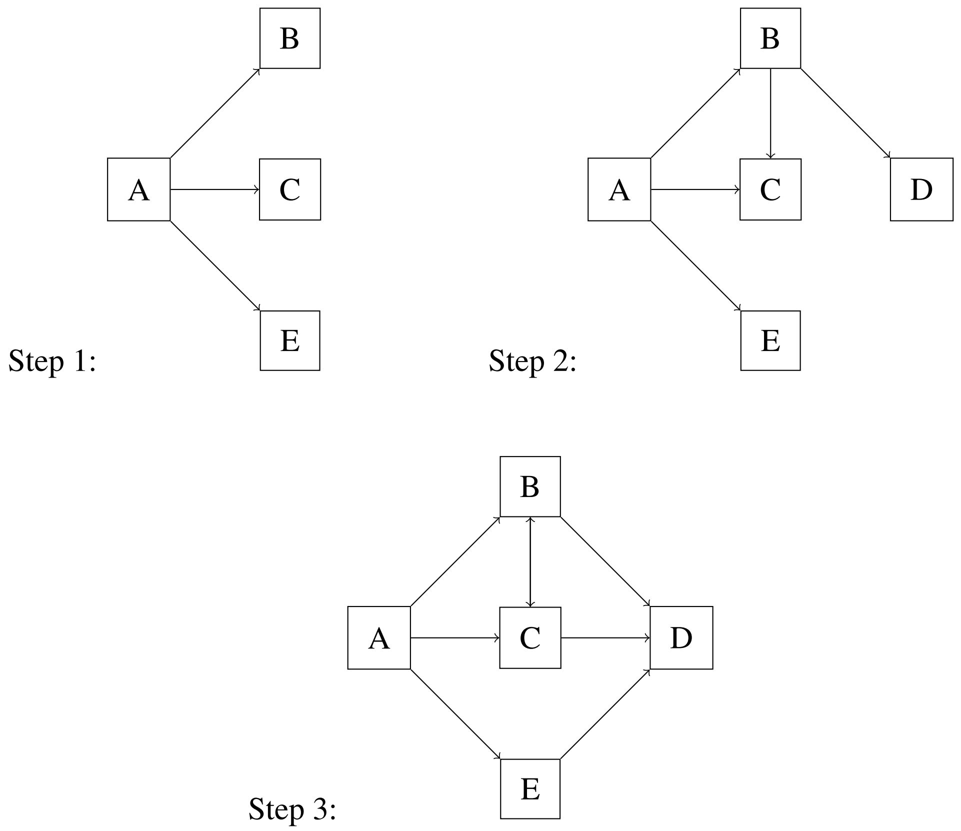


Figure 4: An illustration the creation of a directly follows graph.

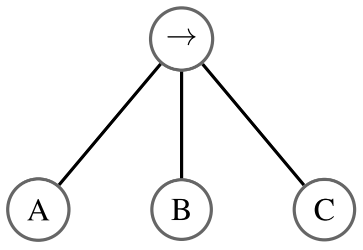


Figure 5: An example of the use of the sequential operator. This process tree shows that activity A occurs, then activity B, and then activity C. In other words, the activities occur from left to right.

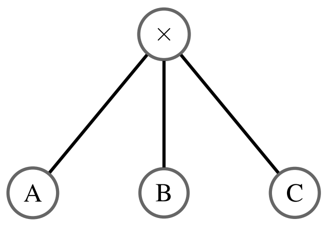


Figure 6: An example of the use of the choice operator. This process tree shows that either activity A, activity B, or activity C occurs.

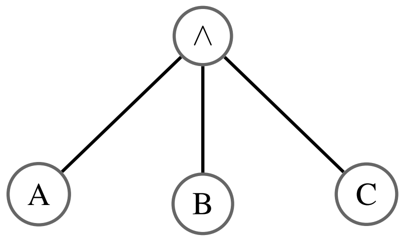


Figure 7: An example of the use of the parallel composition operator. This process tree shows that activity A, activity B, and activity C can be executed at the same time or directly following each other and both activities must be completed before moving to the next phase of the diagram.

parallel composition operator, then A, B, and C all occur, but can occur in any order. For example, A can occur directly before B and then B occurs directly before C, or B can occur directly before A and then A occurs directly before C.

The redo loop operator (⟲) requires a tree with at least two events. There is one "do" event and the rest are "redo" events. The "do" event alternates with the "redo" events, and the process starts and ends with the "do" event (see figure 8). If activity A is the "do" event and activity B and activity C are the "redo" events, an example sequence of events is (A, B, A, C, A, B, A, C, A) [2].

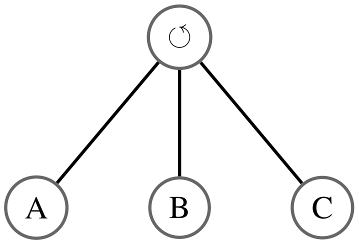


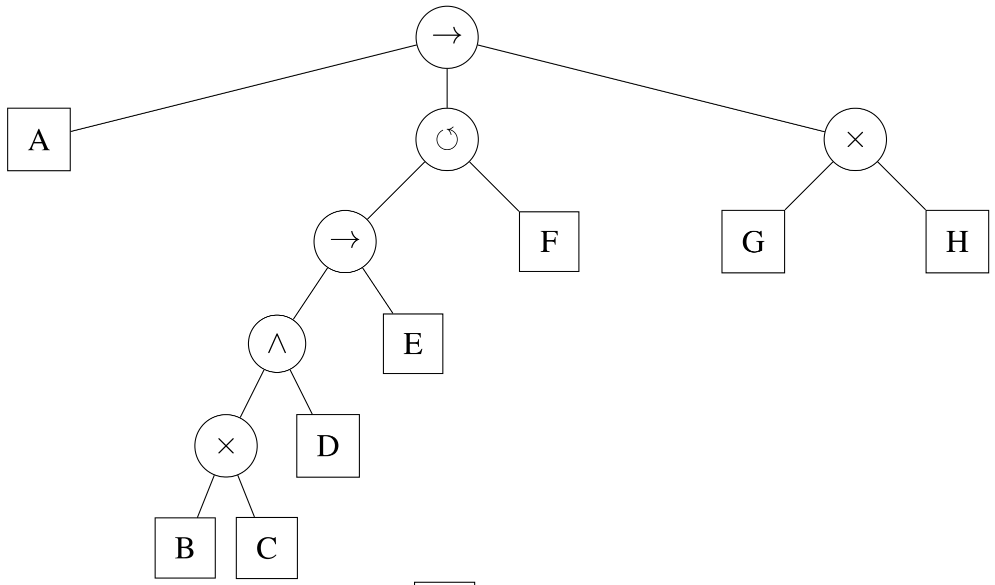
Figure 8: An example of the use of the redo operator. This process tree shows that activity A must occur once. Then, we are either finished or we execute activity B or activity C. If we choose to execute activity B or activity C, then we have to execute activity A again. We can continuously repeat this process until we are done. Some examples of logs generated by the redo operator are (A), (A,B,A,C,A), or (A,B,A).

A large example of a process tree where all of the operators are used is shown in figure 9.

2.5.4 Overview of Inductive Miner

The inductive miner algorithm splits the event log into sublogs. It creates a directly-follows graph from the event logs according to Section 2.5.2. The directly follows graph is a directed graph, where each vertex corresponds to an activity and each directed edge/arc implies that the activity of the vertex where the edge begins occurs before the activity of the vertex where the edge ends.

After the directly-follows graph is created, the algorithm makes cuts to the graph. A cut partitions the set of vertices into distinct sets. There are four different types



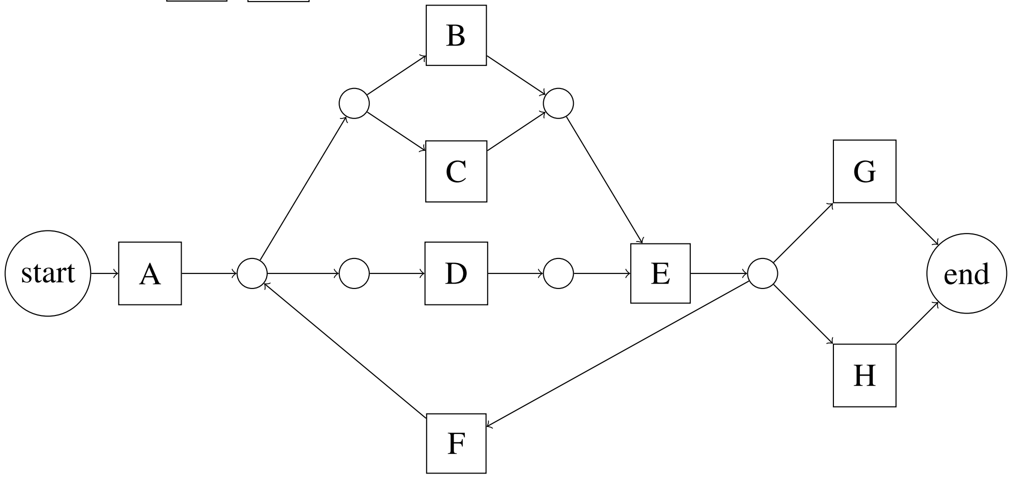


Figure 9: The top figure corresponds to a particular process tree and the bottom figure corresponds to a process map. The process map shows that the process begins with activity A, then proceeds to complete activity D and has a choice of completing activity B or activity C. Once B or C is completed and D is completed, activity E is completed. Then, the process can either go to activity F, activity G, or activity H. If it completes activity H or activity G, then the process is completed. If it completes activity F, then activity D must be completed and activity B or activity C must be completed. The process continues until the end is reached. The process tree shows the same process.

of cuts, each corresponding to one of the four operators that can be present in a process tree: $\rightarrow$, $\times$, $\wedge$, and $\circlearrowleft$. The process operators explain the relationship between activities. If one activity always precedes another they are connected by the $\rightarrow$ operator. If either activity A or activity B occurs, they are connected by the exclusive choice operator. If both activity A and activity B must occur, then they are connected by the parallel composition operator. If activity A can be redone after completing activity B, then they are connected by the redo-loop operator. Therefore, the type of cut explains the relationship between activities.

For each cut, the algorithm looks for exclusive choice cuts ($\times$), then sequence cuts ($\rightarrow$), then parallel choice cuts ($\wedge$), and finally redo-loop cuts ($\circlearrowleft$). The exclusive choice cut is a cut that splits the graph into sets such that there are no arcs going from one set to the other. The sequence cut cuts the graph so that every element in one set comes before every element in the next set. Thus, every element in the previous set occurs before every element in the current set. The parallel choice cut cuts the graph so that every activity in one set is connected to every activity in the other set. The redo loop cut cuts the graph so that one set contains all of the start, or "do", activities and the other sets contain the "redo" activities. The "do" activities start and end the redo loop. The "redo" activities follow an end activity and precede a start activity from the first set.

When deciding which cut to make, the algorithm first looks for a non-trivial exclusive choice cut. Non-trivial means the cut splits the graph into at least two sets. If it finds a non-trivial exclusive choice cut, then it finds and applies a maximal exclusive choice cut. A maximal cut is a cut where no other cut splits the graph into more sets. The maximal cut is a unique cut because if there were multiple distinct maximal cuts, we can take their union to create a cut that partitions the set of vertices into more sets than each of the maximal cuts. This would contradict that those cuts were maximal. Therefore, there is a unique maximal cut. If there are no remaining non-trivial exclusive choice cuts, then the algorithm locates a non-trivial sequence cut and makes a maximal cut. At each step, the algorithm always checks for non-trivial exclusive choice cuts, then non-trivial sequence cuts, then non-trivial parallel choice cuts, and then non-trivial redo loop cuts.

The algorithm continues to make cuts until each activity is isolated. Once each activity is isolated, we know how the activities are related by the operators used to cut the graph. We then can create a process tree from the cut information. From the process tree, we can create a process map as a representation of the process [2].

2.5.5 Inductive Miner: In-Depth Explanation

The inductive miner algorithm begins with an event log. See Figure 10 for the example log we will use in this section. This example log is sorted by Case ID. First, the directly-follows graph is constructed from the event log. Each activity is a node in the graph, and there is a directed edge from activity A to activity B if activity B occurs directly after activity A somewhere in the event log. Throughout this section we will refer to a more complex version of our original morning routine example (Figure 1). The complex morning routine example's event log is shown in Figure 10. For this example, the activities "Wake Up", "Do Hair", "Shower", "Get Dressed", "Brush Teeth", "Leave", "Do Make Up", and "Eat" are all individual nodes in the directly-follows graph. There is an edge from "Wake Up" to "Do Hair" because Jill wakes up and then does her hair. See Figure 11 for the directly-follows graph of the event log in Figure 10.

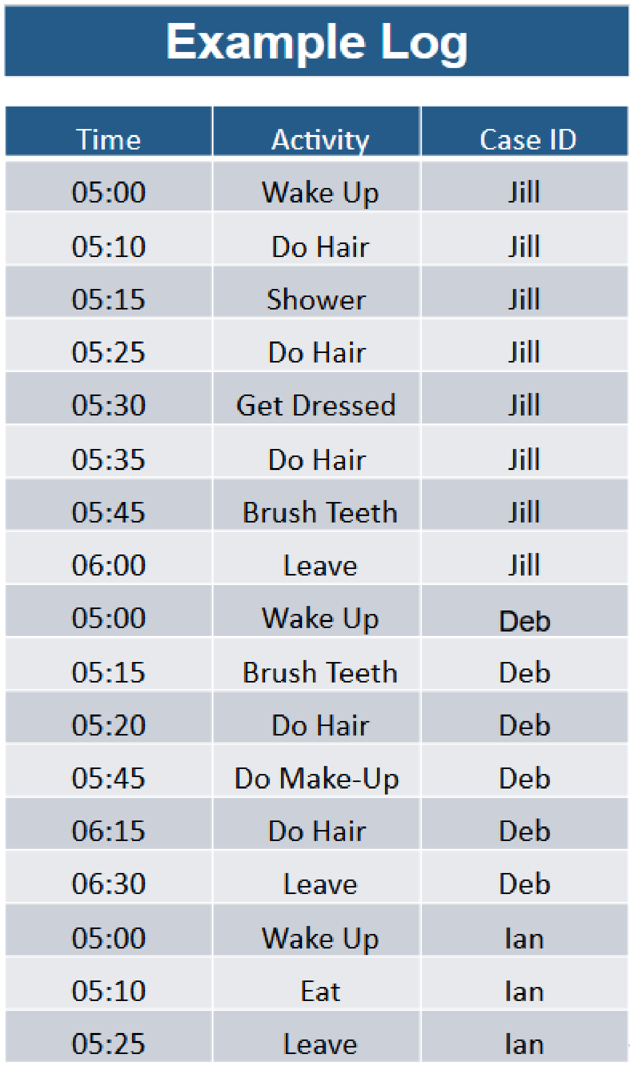
After the creation of the directly-follows graph, the algorithm repeatedly divides the set of nodes into distinct subsets until each subset only consists of one node. The inductive miner chooses how to divide the set by how each node interacts with the other nodes in the directly-follows graph. There are four different types of interactions. Each type of interaction can be classified as an operator on a process tree. The operators are exclusive choice, sequential composition, parallel composition, and the re-do loop. Each division of the set of nodes into subsets is called a cut [2].

Definition (Cut). *Let L be an event log with a directly-follows graph G(L) and the set of all activities in the event log A_L. An n-ary cut of G(L) is a partition of A_L into pairwise disjoint sets A_1, A_2, ..., A_n such that:*

- $A_L = \cup_{i \in \{1,\ldots,n\}} A_i$ *and*
- *For every* $i, j \in \{1, \ldots, n\}$, $i \neq j$, $A_i \cap A_j = \emptyset$

A cut is written as $(\bigoplus, A_1, \ldots, A_n)$ *where* $\bigoplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$

There are four different types of cuts, which are illustrated in Figures 12, 13, 14, and 15. A sequence cut splits the set of nodes based on the directly-follows relation. If there is a path from activity A to activity B, but no path



Example Log

Time	Activity	Case ID
05:00	Wake Up	Jill
05:10	Do Hair	Jill
05:15	Shower	Jill
05:25	Do Hair	Jill
05:30	Get Dressed	Jill
05:35	Do Hair	Jill
05:45	Brush Teeth	Jill
06:00	Leave	Jill
05:00	Wake Up	Deb
05:15	Brush Teeth	Deb
05:20	Do Hair	Deb
05:45	Do Make-Up	Deb
06:15	Do Hair	Deb
06:30	Leave	Deb
05:00	Wake Up	Ian
05:10	Eat	Ian
05:25	Leave	Ian

Figure 10: Example of an event log of three people getting ready in the morning.

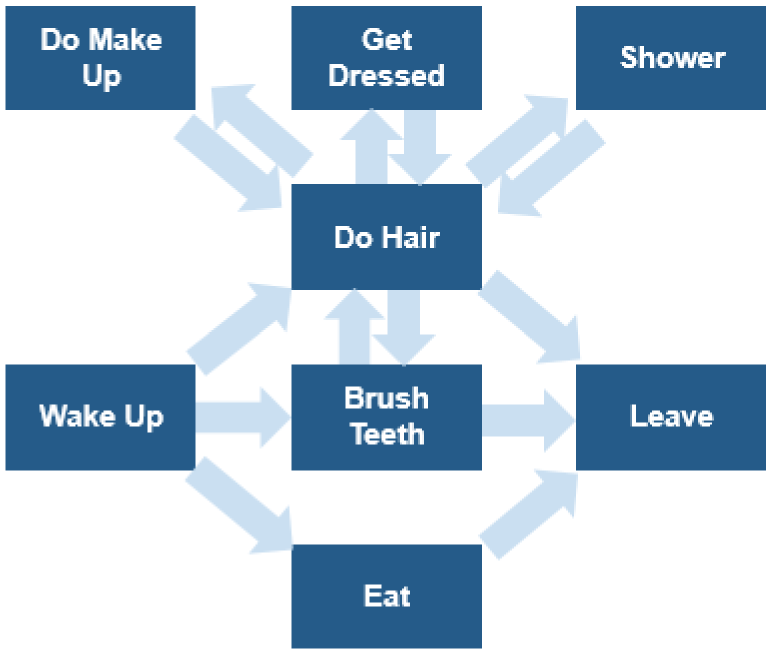


Figure 11: Directly-follows graph created from the event log in Figure 10.

from activity B to activity A, then the sequence cut assigns those activities to different subsets. A path is a sequence of edges that are connected. In our morning routine example in Figure 11, "Wake Up" is followed by "Do Hair", "Brush Teeth", and "Eat", but "Wake Up" does not follow "Do Hair", "Brush Teeth", or "Eat". Thus, the inductive miner assigns the node "Wake Up" to a different subset than "Do Hair", "Brush Teeth", and "Eat". After this sequence cut is made, the set of nodes is split into the sets $A_1 = \{"Wake\ Up"\}$ and $A_2 = \{"Do\ Hair", "Brush\ Teeth", "Do\ Make\ Up", "Get\ Dressed","Shower", "Eat", "Leave"\}$ [2].

Definition (Sequence Cut)**.** *A sequence cut is a cut* $(\rightarrow, A_1, ..., A_n)$ *such that for all i,j in* $\{1,...,n\}$ *and for all a in* A_i *and b in* A_j *with* $i < j$, *we have* $(a \mapsto_L^+ b) and (b \not\mapsto_L^+ a)$.

An exclusive-choice cut splits nodes into different subsets if they are not connected by an edge. In our morning routine example, "Eat" and "Brush Teeth" are not connected by an edge, so an exclusive choice cut assigns "Eat" and "Brush Teeth" to different subsets. Since "Eat" and "Brush Teeth" are split by the exclusive choice cut in the morning routine event log, a person either brushes their teeth or eats.

Definition (Exclusive Choice Cut)**.** *An exclusive choice cut is a cut* $(\times, A_1, ..., A_n)$ *such that for every* $i, j \in \{1,...,n\}$, *for every* $a \in A_i$, *and for every* $b \in A_j$ *with* $i \neq j$, *we have* $a \not\mapsto_L b$.

A parallel cut splits activities that occur in parallel. In other words, if activity A is split from activity B by a parallel cut, then when activity A occurs in the event log activity B follows, and when activity B occurs in the event log, activity A follows. Both activities must be executed before moving on to the next activity in the model. In order to execute this kind of cut, each newly created subset must contain a start and an end event to ensure that the activities are related by the parallel composition operator and not the redo-loop operator. In our morning routine example, a parallel cut splits "Do Hair" and "Brush Teeth" because in the event log, if the person chooses to brush their teeth or do their hair, they always also do their hair or brush their teeth respectively [2].

Definition (Parallel Cut)**.** *A parallel cut is a cut* $(\wedge, A_1, ..., A_n)$ *such that*

- *For every* $i \in \{1,...,n\}$ $(A_i \cap A_L^{start} \neq \emptyset) \wedge (A_i \cap A_L^{end} \neq \emptyset)$ *where* A_L^{start} *is the set of start events and* A_L^{end}*is the set of end events, and*
- *For every* $i, j \in \{1,...,n\}$*, for every* $a \in A_i$*, and for every* $b \in A_j$ *with* $i \neq j$ *and we have* $a \mapsto_L b$ *.*

A redo-loop cut corresponds to the redo-loop operator. The cut splits the "do" activity and each of its "redo" activities into separate subsets. Recall that the "do" activity is the activity executed first and last in the redo loop, and the "redo" activities are the activities that can be executed in order to redo the "do" activity. In order to execute a redo-loop cut, all of the start and end events, along with the "do" event must be in the same subset. Every event in the subsequent subsets should have an edge between a start event and an end event in the first subset. There should be no edges between these subsequent subsets. In other words, there should only be edges between the first subset and all of the subsequent subsets. In our morning routine example, the "Do Hair" activity is the "do" activity, and the "redo" activities are "Do Make Up", "Get Dressed", and "Shower". After the execution of a redo-loop cut, "Do Hair" is in the first subset, and "Do Make Up", "Get Dressed", and "Shower" are each in their own subsequent subsets [2].

Definition (Redo-loop Cut)**.** *A redo loop cut is a cut* $(\circlearrowleft,\ A_1,\ ...,\ A_n)$ *such that all of the following hold true:*

- $n \geq 2$,
- $A_L^{start} \cup A_L^{end} \subseteq A_1$,
- $\{a \in A_1 | \exists_{i \in \{2,...,n\}} \exists_{b \in A_i} a \mapsto_L^+ b\} \subseteq A_L^{end}$,
- $\{a \in A_1 | \exists_{i \in \{2,...,n\}} \exists_{b \in A_i} b \mapsto_L^+ a\} \subseteq A_L^{start}$,
- *For all* $i, j \in \{2,...,n\}, for\ all\ a \in A_i, for\ all\ b \in A_j, i \neq j \Rightarrow a \not\mapsto_L b$,
- *For all* $i \in \{2,...,n\}, for\ all\ b \in A_i,\ \exists_{a \in A_L^{end}} a \mapsto_L b \implies for\ \ all\ a' \in A_L^{end}\ a' \rightarrow_L b$*, and*
- *For all* $i \in \{2,...,n\}, for\ all\ b \in A_i,\ \exists_{a \in A_L^{start}} b \mapsto_L a \implies\ for\ all\ a' \in A_L^{start}\ b \rightarrow_L a'$.

The algorithm prioritizes each type of cut in the following order: exclusive choice cuts, sequence cuts, parallel cuts, and then redo-loop cuts. Therefore, each time the algorithm attempts to make a cut, it first looks for an exclusive choice cut. The algorithm always makes the maximal cut, which means it makes the cut that will divide the set into the most subsets. As previously stated, the maximal cut is a unique cut. It continues to search for and make cuts until each activity is in its own subset. Once each activity is in its own subset, we know how all of the activities are related. Since each cut corresponds to a process operator, the cuts the algorithm makes to separate activities explains how the activities are related. Once we know which operators connect the activities, we are able to use the type of cuts necessary to split the graph to create a process tree. For example, we know that a sequence cut corresponds to the sequential operator. A sequence cut splits "Wake-Up" from the rest of the activities. Therefore, we know that "Wake-Up" occurs before all of the other activities. We repeatedly use the cut information to create the process tree of the event log [2].

We will now revisit our complex morning routine example from Figure 10 and perform the Inductive Miner. First, we create the directly-follows graph in Figure 11. Then, we use this graph to make cuts. See Figures 12, 13, 14, and 15 for illustrations of all of the cuts. There are no exclusive choice cuts in the graph, so we look for a sequence cut. We find that we can make a sequence cut such that $A_1 = \{"Wake\,Up"\}$, $A_2 = \{"Do\,Make\,Up", "Get\,Dressed", "Shower", "Do\,Hair", "Brush\,Teeth", "Eat", "Leave"\}$. Now, we look for a maximal sequence cut, and we find that it is $A_1 = \{"Wake\,Up"\}$, $A_2 = \{"Leave"\}$, and $A_3 = \{"Do\,Make\,Up", "Get\,Dressed", "Shower", "Do\,Hair", "Brush\,Teeth", "Eat"\}$. This is the maximal cut because it splits A_L into the greatest number of subsets. This cut is shown in Figure 12. Now, A_1 and A_2 only consist of single activities, so we only need to look at splitting A_3 into more subsets [2].

We first look for a maximal exclusive choice cut, and we find that A_3 has an exclusive choice cut that splits A_2 into 2 subsets. We check that this is the maximal cut, and it is. Now, we have $A_1 = \{"Wake\,Up"\}$, $A_2 = \{"Leave"\}$, $A_3 = \{"Eat"\}$, and $A_4 = \{"Brush\,Teeth", "Do\,Hair", "Do\,Make\,Up", "Get\,Dressed", "Shower"\}$. Now, A_1, A_2, and A_3 only consist of isolated activities, so we only need to make further cuts to A_4. This cut is shown in Figure 13 [2].

Next, we look for cuts in A_4. There are no exclusive choice cuts or sequence cuts, so we look for a parallel cut. It appears as though the following

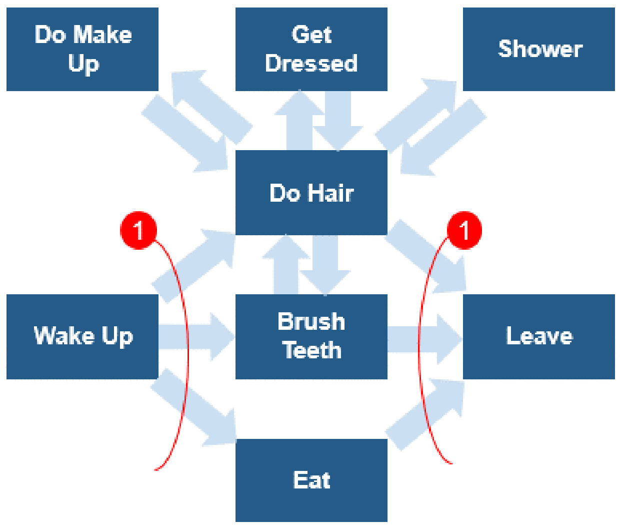


Figure 12: The first cut made to the directly-follows graph in the morning routine example from the inductive miner algorithm. It is a sequence cut.

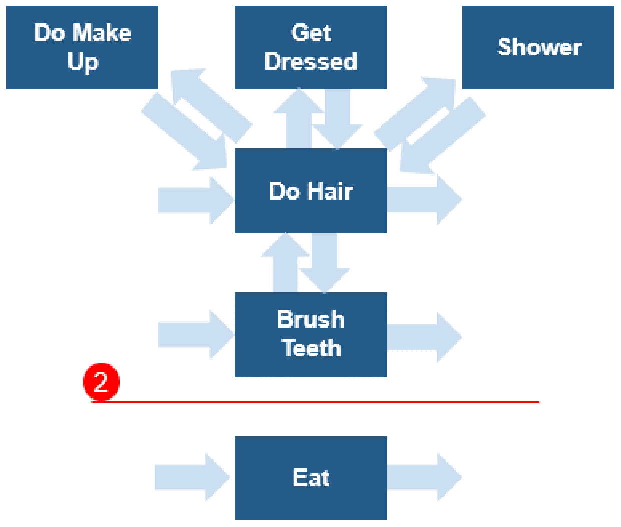


Figure 13: The second cut made to the directly-follows graph in the morning routine example from the inductive miner algorithm. It is an exclusive choice cut.

split could be a parallel cut that splits A_4 into $A_4 = \{$"*Brush Teeth*"$\}$ and $A_5 = \{$"*Do Hair*", "*Get Dressed*", "*Do Make Up*", "*Get Dressed*", "*Shower*"$\}$. We check the requirements. It meets the first requirement because $A_L^{start} = \{$"*Do Hair*", "*Brush Teeth*"$\}$ and $A_L^{end} = \{$"*Do Hair*", "*Brush Teeth*"$\}$. So, $(A_4 \cap A_L^{start}) = \{$"*Brush Teeth*"$\} \neq \emptyset$ and $(A_5 \cap A_L^{start}) = \{$"*Do Hair*"$\} \neq \emptyset$. Also, $(A_4 \cap A_L^{end}) = \{$"*Brush Teeth*"$\} \neq \emptyset$ and $(A_5 \cap A_L^{end}) = \{$"*Do Hair*"$\} \neq \emptyset$. The cut does not meet the second requirement of for every $i, j \in \{1, ..., n\}$, for every $a \in A_i$, and for every $b \in A_j$ with $i \neq j$ $a \mapsto_L b$ because every element in A_5 is not directly followed by every element in A_4. Therefore, we look for a redo loop cut. We find a maximal redo loop cut that splits A_4 into 4 subsets. $A_4 = \{$"*Shower*"$\}$, $A_5 = \{$"*Do Make Up*"$\}$, $A_6 = \{$"*GetDressed*"$\}$, and $A_7 = \{$"*Do Hair*", "*Brush Teeth*"$\}$. It meets the requirements for a redo-loop because:

- $n = 5 \geq 2$,
- $A_L^{start} \cup A_L^{end} = \{$"*Brush Teeth*", "*Do Hair*"$\} \subseteq A_7$,
- $\{a \in A_7 \mid \exists_{\, i \,\in\, \{4,\, 5,\, 6\}} \exists_{\, b \,\in\, A_i}\; a \mapsto_L^+ b\} \subseteq A_L^{end}$,
  There is a path from "*Do Hair*" and a path from "*Brush Teeth*" to an element in A_4, A_5, and A_6.
- $\{a \in A_7 \mid \exists_{i \,\in\, \{4,\, 5,\, 6\}} \exists_{b \,\in\, A_i}\; b \mapsto_L^+ a\} \subseteq A_L^{start}$,
  There is a path from an element in A_4, A_5, and A_6 to "*Do Hair*" and to "*Brush Teeth*".
- $For\ all\ i, j \in \{4,5,6\}, for\ all\ a \in A_i, for\ all\ b \in A_j, i \neq j \Rightarrow a \not\mapsto_L b$,
  There are no edges between elements in A_4, A_5, and A_6.
- $For\ all\ i \in \{4,5,6\}, for\ all\ b \in A_i,\ \exists_{\, a \,\in\, A_L^{end}}\ a \mapsto_L b \implies for\ all\ a' \in A_L^{end}\ a' \rightarrow_L b$,
  Each element in A_4, A_5, and A_6 is preceded by an end event.
- $For\ all\ i \in \{4,5,6\}, for\ all\ b \in A_i\ \exists_{\, a \,\in\, A_L^{start}} b \mapsto_L a \implies for\ all\ a' \in A_L^{start}\ b \rightarrow_L a'$.
  Each element in A_4, A_5, and A_6 is followed by a start event.

The redo-loop cut is shown in Figure 14 as cut 3. Since A_4, A_5, and A_6 all consist of isolated events, we now only need to focus on A_7 [2].

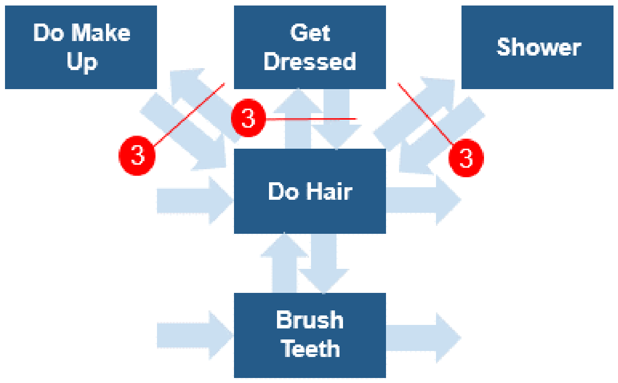


Figure 14: The third cut made to the directly-follows graph in the morning routine example from the inductive miner algorithm. It is a redo-loop cut.

There are no exclusive choice cuts or sequence cuts, so we look for a parallel cut. We find the parallel cut $A_7 = \{"Do\ Hair"\}$ and $A_8 = \{"Brush\ Teeth"\}$. This meets the first requirement of the parallel cut because $A_L^{start} = A_L^{end} = \{"Brush\ Teeth", "Do\ Hair"\}$, $(A_7 \cap A_L^{end}) = (A_7 \cap A_L^{start}) = \{"Do\ Hair"\} \neq \emptyset$, and $(A_8 \cap A_L^{end}) = (A_8 \cap A_L^{start}) = \{"Brush\ Teeth"\} \neq \emptyset$. This meets the second requirement because every element in A_7 is directly followed by every element in A_8 and vice versa. This parallel cut is shown in Figure 15. After this cut, each activity is isolated in its own subset, and we are done cutting the graph [2].

Each cut tells us how the activities are related, so we are able to construct the process tree in Figure 16. The first cut we made told us that "Wake Up" occurs first then the events in the middle of the directly-follows graph, and then "Leave". This is demonstrated in the process tree by the sequential operator as the top node, "Wake Up" as its leftmost child, "Leave" as its rightmost child, and all other events in the middle. Next, we have the exclusive choice cut, which is shown as the sequential operator node's middle child. This cut separated "Eat" from the rest of the nodes, which is shown by "Eat" being the only child on the right of the exclusive choice operator. This process continues to build the process tree of our event log. Once each cut is represented in the process tree, the algorithm finishes [2].

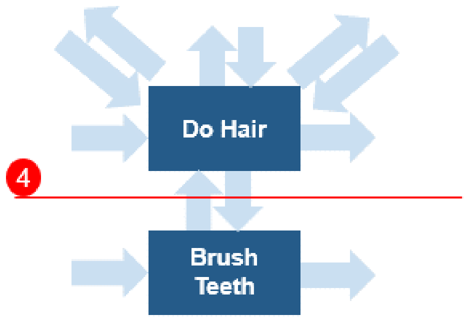


Figure 15: The fourth cut made to the directly-follows graph in the morning routine example from the inductive miner algorithm. It is a parallel cut.

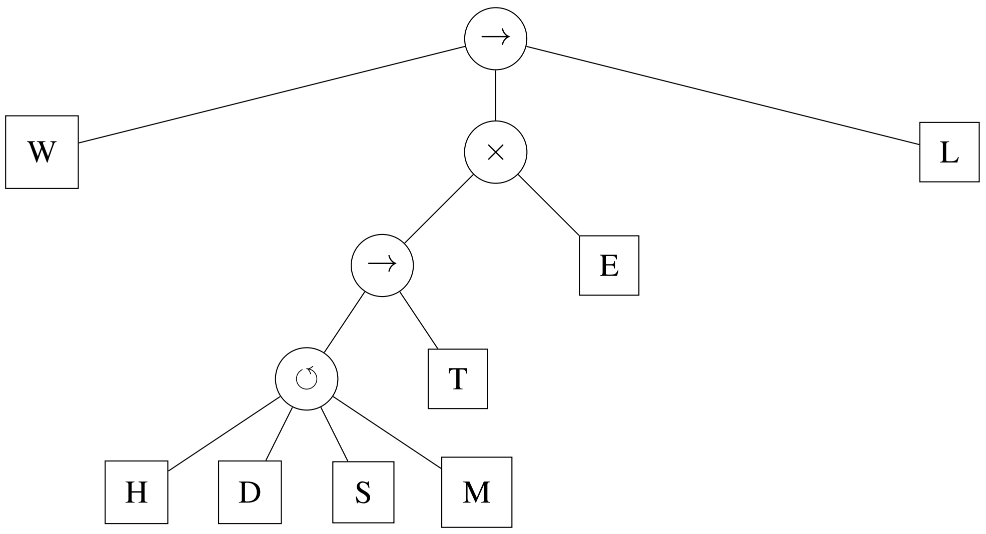


Figure 16: The process tree developed from the cuts made by the inductive miner algorithm to the directly-follows graph, as shown in Figures 12, 13, 14, and 15. The "Wake Up" activity is represented by the *W* square. "Eat" is represented by the *E* square. "Brush Teeth" is represented by the *T* square. "Do Hair" is represented by the *H* square. "Get Dressed" is represented by the *D* square. "Shower" is represented by the *S* square. "Do Make Up" is represented by the *M* square. "Leave" is represented by the *L* square.

3 Methodology

Information from this section has been redacted due to the containment of Credit Suisse proprietary information. We model Credit Suisse's advisory process from the user click data. Credit Suisse has five steps that its employees follow for the advisory process: "Needs Analysis", "Financial Concept", "Client Profile", "Strategy", and "Implementation", so we check that the user data confirms the conformance to that predefined five-step model. We format the data so that they can be imported to ProM using Python, import the data to ProM, run the Inductive Miner algorithm on the event logs, and analyze the outputted model.

3.1 Identifying a Data Set

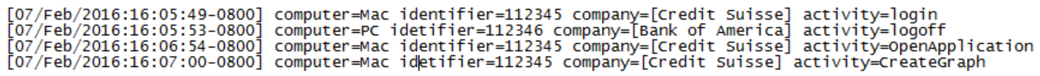
In order to begin process mining, we must have a clearly defined data set. This data set must represent a process. If there is no underlying process, then the output from the model will be invalid because the model will display steps in a process that does not exist. A partial model can be derived from event logs that only consist of part of the process. This partial model is still valid, but it is missing some activities. In order to get valid results, we must narrow the scope of the data to only include one process but also have a large enough scope that includes every activity in the process.

Credit Suisse records every click a user makes in its applications. For example, when an employee clicks a link, the Credit Suisse system logs that employee ID, a description of the click, and the time. Our data set is user click data. User click data for every event consists of a timestamp, a description of the click, an anonymous identifier of the user, and other fields. We refer to the description of the click as the name of the click. We are only concerned with the timestamp, the name of the click, and the identifier of the user. This identifier is only used to construct cases, no client or employee identifying information was used. The name of the click is the activity and the anonymous identifier of the user is the case ID. An example of click data is *10:00, Needs Analysis, 0001*, where the timestamp is 10:00, the click name is Needs Analysis, and the identifier is 0001. The click names correspond to steps in the advisory process, so we can use them to derive a process model. The click data includes other extraneous information, such as location information, that we discard. The click data also includes click names that are not related to the advisory process that must be discarded for our analysis. Thus, we must format the data.

3.2 Formatting the Data

The raw log files contain more information than we need and their format is not compatible with ProM. We use a Python script to write the necessary fields to a new file in CSV format. CSV stands for Comma Separated Values, so each field is separated by a comma. ProM accepts a CSV file with the field names separated by commas at the top, and then the data listed below. Each line in the file corresponds to a specific event. In order to perform the analysis, each event must consist of an identifier for each login session, a timestamp, and an activity description.

In our raw data file, each line refers to an event, and the fields are delineated by spaces. See example in figure 17.



```
[07/Feb/2016:16:05:49-0800] computer=Mac identifier=112345 company=[Credit Suisse] activity=login
[07/Feb/2016:16:05:53-0800] computer=PC idetifier=112346 company=[Bank of America] activity=logoff
[07/Feb/2016:16:06:54-0800] computer=Mac identifier=112345 company=[Credit Suisse] activity=OpenApplication
[07/Feb/2016:16:07:00-0800] computer=Mac idetifier=112345 company=[Credit Suisse] activity=CreateGraph
```

Figure 17: A sample of how the raw data is formatted.

Our analysis only requires the timestamp, the case ID, and the activity. The company and computer are extraneous fields. Notice that some fields, like the company, can contain spaces. Within our file, fields are separated by spaces. Words within fields are separated by spaces as well. This complicates the data extraction. We handle this by looking at each line of the event log individually. We use the Python split line method to turn the line into a list, where the start and end of an item in the list is denoted by a space. For example, if we use the split line method on the first line in Figure 17, our list would be [07/Feb/2016:16:05:49-0800], computer=Mac, identifier=112345, company=[Credit, Suisse], activity=login. We refer to the previous list as list1. If we use the split line method for line 2 in Figure 17, our list would be [07/Feb/2016:16:05:53-0800], computer=PC, identifier=112346, company=[Bank, of, America], activity=logoff. We refer to this list as list2. The lengths of these lists are different due to the use of spaces within fields. Therefore, the fields occur at different places in our lists. In Python, the items in a list can be accessed according to their location in the list. For example, list1[5] is activity=login, while list2[6] is activity=logoff. If our fields were all in the same location of their respective list, we could simply access each entry according to their number. Another problem with this format is that fields could be split between two different entries in the list. However, all of our required fields do not contain spaces, so we do not need to address this problem.

Since the fields we need to extract are located in different spots in the each line's list, we use the string comparison to search for each field name within the list. Once the field name is located, we remove the field name and put the contents of the field in a new list. For example, if we search list1 for the substring "identifier=", we find the entry identifier=112345. Following this, the string is partitioned to include just the relevant information; say perhaps we were only interested in the "123", we find and remove the characters before the "123" and after. Then, we put the resulting "123" into the new list. We continue to find our next field in the list and add the contents of the field to the list. We repeat this until all of our fields have been located and added to the list. Finally, we write the list to the CSV file and repeat the process with the next line in the raw data file. After this has been completed for each line, we have correctly formatted our data to import into ProM.

3.3 Analysis in ProM

Once the data is formatted as a CSV, we can import the data into ProM. From there we decide which algorithm to use to create our process model. We choose to use the inductive miner because the algorithm is sound and known for working well on real life data. The inductive miner also identifies deviations from the process model and includes statistics, such as the time it takes to complete an activity. All of these capabilities can help to improve efficiency and eliminate redundancy in the advisory process [5]. The inductive miner also documents the mainstream behavior of a process, which is what Credit Suisse requested.

3.4 Analysis with the Advisory Process Team

After our initial analysis in ProM, we approach the advisory process team to gain insight on what our model showed compared to what they expected. We show the bottlenecks and deviations in the advisory process to the advisory team in search of known explanations. If there are no known explanations, the advisory team uses the information to improve the process and identifies areas in the model for us to analyze further. If all deviations are explained, we have confirmed that the advisory process runs the way it is expected.

4 Results

Information in this section is redacted due to the containment of Credit Suisse proprietary information. Our original data set consisted of one week of click data from all of the clicks in the application. Initially we expected to derive an overall process model for how users interact with the application. However, it is not possible to derive a process model from something that does not include a process to begin with. The clicks represent what each user does while they are logged in to the system. Every user behaves differently, thus the combined event logs produced a nonsensical model where each activity could loop back to itself or any other activity in the diagram. We realized that we must narrow our scope with a new script focusing on only the events involved in a single process. We choose to focus on the advisory process.

Upon executing our second parsing script, we are able to visualize a specific subset of the log data focusing on Credit Suisse's advisory process. The Inductive Miner creates a process model based on the log data, which is shown in Figure 18. From the resulting diagram, we identify the bottlenecks in their process, as well as less used applications. Each edge in the diagram shows the number of times it is traversed, so edges with fewer traversals are used less frequently. The model outputs the time to complete each activity, so the user can determine the activities that consume the most time. For example, the "Service Profile" activity takes much longer than any other activity, and the relations activity only occurs 54 times. There were no deviations in the model, which implies that users are behaving similarly.

After consulting with the advisory team, we realized we needed to analyze a longer time frame. The original data set did not have a long enough time frame, thus did not include a significant initial event of the advisory process, a questionnaire that the client fills out. The second set was of a longer time frame and did include the missing initial event from the first set. See Figure 19 for the second data set's process model.

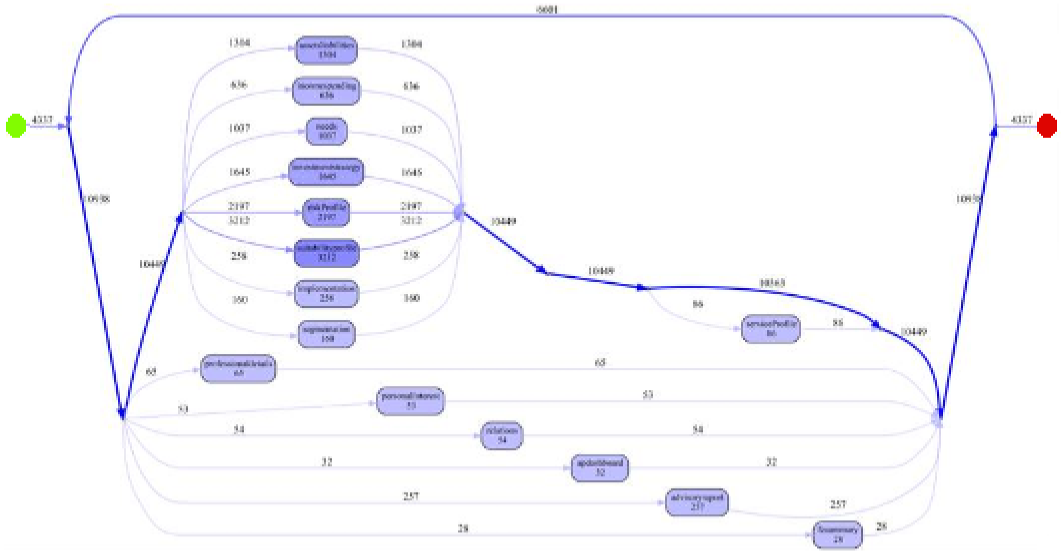


Figure 18: The process model for 1 week of user click data derived from the inductive miner algorithm.

In each model, we see a start point (represented by a green dot) and an end point (represented by a red dot), with the steps of the process in between. Each model includes the number of times an edge is traveled in between activities. The thicker, darker lined edges are the edges that are most traveled in the process. Each node in the models represents an activity in the advisory process. Figure 19 includes circles which represent splits in the process. The circles with an X are an "or" split, which means the process can choose one of the edges to follow that is connected to the "or" split. The circles without an X are "and" splits, where all paths connected to the "and" split must be followed.

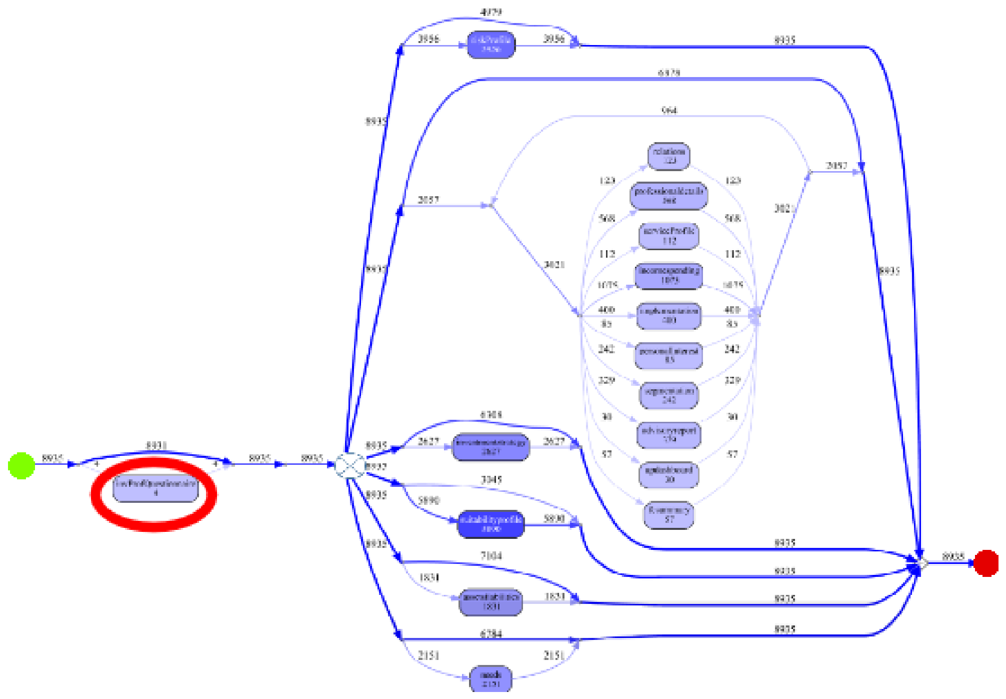


Figure 19: The process model for 2 weeks of user click data derived from the inductive miner algorithm. The activity circled in red is the activity that did not appear in our original model. This shows that we did not have the correct time span for our original model.

5 Conclusion

Our project successfully demonstrated that process mining can be used to reduce a business' operational costs by identifying redundant tasks and bottlenecks in their processes. For example, we found that the "Service Profile" activity was significantly slower than other activities. We used user click data to create a process model and identify deviations and inefficiencies in Credit Suisse's advisory process. In the future, we recommend that Credit Suisse uses process mining to create process models and determine usage statistics and capabilities for other processes within their organization, such as the Loan Approval process. By utilizing process mining on their processes, Credit Suisse will gain a better understanding of how their processes actually run and can make their processes more efficient by eliminating any redundant tasks and bottlenecks.

In order to optimally analyze individual processes, we recommend Credit Suisse to look at data over long periods of time. We analyzed two sets of data for our study of the advisory process, one for one week of click data and the other for two weeks. The model of our second data set included an activity that did not appear in the model of the first data set because some activities are executed at the end of the year and some are executed at the end of the quarter. From the process perspective, both graphs are similar, but they are formatted differently in Figures 18 and 19. If Credit Suisse creates process models from data sets for long periods of time, then their resulting process models will be more accurate and will include information that may be left out from a data set of smaller scope.

We also recommend consulting with the advisory team for more insight into the advisory process. They may be aware of certain bottlenecks in the process, such as the "Service Profile" activity which takes much longer than any other activity, and have an explanation for why it takes so much time to complete. We recommend consulting with the team in charge of every other process that Credit Suisse chooses to model in order to gain a better understanding of the process and the process model.

References

[1] Credit Suisse, "Our company." Retrieved from: https://www.credit-suisse.com/ch/en/about-us/our-company.html.

[2] W. van der Aalst, *Process Mining Data Science in Action*. Springer-Verlag Berlin Heidelberg, 2016.

[3] Credit Suisse, "Advisory process." Retrieved from: https://www.credit-suisse.com/nl/en/private-banking/advisory-process.html.

[4] W. van der Aalst, "Process mining: Data science in action." Coursera [Online Course], 2016. Retrieved from: https://www.coursera.org/learn/process-mining/home/welcome.

[5] J. Buijs, "Introduction to process mining with prom." Future Learn [Online Course], 2016. Retrieved from: https://www.futurelearn.com/courses/process-mining.