UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Volodymyr Leno

Incremental Discovery of Process Maps

Master's Thesis (30 ECTS)

Supervisor(s): Marlon Dumas,
Michal Rosik

Tartu 2017

Incremental Discovery of Process Maps

Abstract:

Process mining is a body of methods to analyze event logs produced during the execution of business processes in order to extract insights for their improvement. A family of process mining methods, known as automated process discovery, allows analysts to extract business process models from event logs. Traditional automated process discovery methods are intended to be used in an offline setting, meaning that the process model is extracted from a snapshot of an event log stored in its entirety. In some scenarios however, events keep coming with a high arrival rate to the extent that it is impractical to store the entire event log and to continuously re-discover a process model from scratch. Such scenarios require online automated process discovery approaches. Given an event stream produced by the execution of a business process, the goal of an online automated process discovery method is to maintain a continuously updated model of the process with a bounded amount of memory while at the same time achieving similar accuracy as offline methods. Existing automated discovery approaches require relatively large amounts of memory to achieve levels of accuracy comparable to that of offline methods. This thesis proposes a online process discovery framework that addresses this limitation by mapping the problem of online process discovery to that of cache memory management, and applying well-known cache replacement policies to the problem of online process discovery. The proposed framework has been implemented in .NET, integrated with the Minit process mining tool and comparatively evaluated against an existing baseline, using real-life datasets.

Keywords:

Process Mining, Process Discovery, Event Stream, Process Map, Cache Replacement Policy

CERCS: P170 – Computer science, numerical analysis, systems, control

Kuhjuv protsessikaartide avastamine

Lühikokkuvõte:

Protsessikaeve on meetodite kogu, analüüsimaks, protsesside teostuse jooksul loodud, sündmuste logisid, et saada teavet nende parandamiseks. Protsessikaeve meetodite kogu, mida nimetatakse automatiseeritud protsessi avastuseks, lubab analüütikutel leida informatsiooni äriprotsesside mudelite kohta sündmuste logidest. automatiseeritud protsessi avastus meetodeid kasutatakse tavaliselt ühenduseta keskkonnas, mis tähendab, et protsessi mudel avastatakse hetketõmmisena tervest sündmuste logist. Samas on olukordi, kus uued juhtumid tulevad peale sellise suure kiirusega, et ei ole mõtet salvestada tervet sündmuste logi ja pidevalt nullist taasavastada mudelit. Selliste olukordade jaoks oleks vaja võrgus olevaid protsessi avastus meetmeid. Andes sisendiks protsessi teostuse käigus loodud sündmuste voo, võrgus oleva protsessi avastus meetodi eesmärk on järjepidevalt uuendada protsessi mudelit, tehes seda piiratud hulga mäluga ja säilitades sama täpsust, mida suudavad meetodid ühenduseta keskkondades. Olemas olevad meetodid vajavad palju mälu, et saavutada tulemusi, mis oleks võrreldavad ühenduseta keskkonnas saadud tulemustega. Käesolev lõputöö pakub välja võrgus oleva protsessi avastusraamistiku, ühtlustades protsessi avastus probleemi vähemälu haldusega ja kasutades vähemälu asenduspoliitikaid lahendamaks antud probleemi. Loodud raamistik on kirjutatud kasutades .NET-i, integreeritud Minit protsessikaeve tööriistaga ja analüüsitud kasutades elulisi ärijuhte.

Võtmesõnad:

Protsessi Kaevandamine, Protsess Avastus, Juhtum Vool, Protsessi Kaardid

CERCS: P170 – Computer science, numerical analysis, systems, control

Table of Contents

List of Abbreviations

The table below describes the meaning of the various abbreviations and acronyms used in the thesis.

Abbreviation	Meaning
AMQP	Advanced Message Queuing Protocol
BPMN	Business Process Model and Notation
CRM	Customer Relationship Management
EPC	Event-driven Process Chain
ERP	Enterprise Resource Planning
HTTPS	HyperText Transfer Protocol Secure
SAS	Shared Access Signature
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Level Security
WFM	Workforce Management
YAWL	Yet Another Workflow Language

1 Introduction

The following chapter aims to give a brief overview of the problem and identify research questions with the thesis main goals. In addition, it provides the structure description of the paper.

1.1 Problem Statement

Business processes take an important part in modern organizations, describing the list of activities that have to be done, their execution order and performers. Nowadays, many processes are either supported or monitored by information systems (e.g., ERP, CRM, WFM systems). The information about the execution of these processes is stored in files called event logs. An event log consists of a set of traces, each representing one possible process execution. A trace consists of a set of events, where an event contains a timestamp and an event type, which refers to an activity of the process (e.g. “Assess Claim”, “Notify Customer”, etc.). Process discovery aims to construct a process model from an event log. Different process discovery techniques support different notations (e.g., Petri Net, BPMN, YAWL and EPC). In this thesis, we are specifically interested in process discovery techniques that produce a process map as output. In this context, process map is a directed graph where each node corresponds to an event type and each edge represents a direct-follows relation, which exists between two event types (e.g. between “Receive Order” and “Check Order”) if there is at least one trace where the first of these events immediately preceded the second one. By representing the process in a graphical way, process discovery helps to understand and analyze the process in order to identify problems and bottlenecks. Based on the results of analysis process improvement can be done.
However, the amount of data is continuously increasing, systems produce large and complex datasets that are hard to manage and store. This causes the computational and memory issues. One of the solutions to this problem is to process events immediately when they occur in order to enable real-time analysis. This approach is known as online process discovery or incremental process discovery.
There are many process discovery algorithms, but all of them designed for offline processing, taking as input complete event logs. However, it may be impossible to store all events when working with event streams. Thus, these algorithms fail in this case and tend to generate unreliable and outdated results.
Moreover, some processes are changing in time. They might change due to periodic changes (e.g., in the different time of the year, there are different sales rates or market demands), or thanks to changing conditions (e.g., changes in the strategy of the company, market changes). These changes affect the processes and it is important to detect them and update the model correspondingly.

In order to be able to work with event streams, the process discovery algorithms have to meet the following requirements [1]:

- It is impossible to store the complete stream
- It is impossible to process data more than one time (backtracking is not feasible)
- Data model has to be quickly updated
- Algorithm has to cope with variable system conditions

Currently, only a few algorithms can deal with the event streams. All of them are modifications and extensions of existing offline algorithms.

In addition, Process Discovery itself faces challenges such as dealing with noise, incompleteness, balancing overfitting and underfitting, representational bias limitations [2].

The core problem to be addressed in the thesis can be formulated as:

"Given a process map PM computed from an event log at time T and given the set of events ES that has occurred between time T and T' we need to compute the process map PM' at time T'."

The main research question is:

How to perform process discovery on the event streams?

The other research questions are following:

1) How to mine data immediately when it occurs?
2) How to continuously update the model in a computationally efficient way?
3) How to deal with concept drifts?
4) How to deal with process discovery problems?

1.2 Contribution

The main goal of this work is to introduce an efficient process discovery algorithm in order to cope with event streams, which will be built in the client application with an intuitive and user-friendly interface. The thesis aims at extending the process map construction technique implemented in the Minit tool to handle event streams, therefore the main focus will be done on process maps discovery.
In the thesis, the current state of incremental process discovery problem will be studied and existing solutions will be analyzed. Based on the results of analysis, an efficient algorithm will be presented and developed.

1.3 Structure Description

The thesis is organized as follows: Section 2 presents the basic concepts of process mining and Section 3 provides the overview of state of the art of the problem with analysis of existing solutions; in Section 4 the new solution based on the Lossy Counting with Budget algorithm is introduced and the implementation details are reported; Section 5 presents the results of empirical evaluation and Section 6 concludes the paper.

2 Background

This chapter aims to give basics about process mining and process discovery. In addition, it contains the overview of process maps and data/event streams.

2.1 Introduction to process discovery

Process Mining is relatively young research discipline, which takes place between machine learning and data mining from the one hand and process modeling and analysis from the other hand [2]. It aims to discover, monitor and improve the processes, extracting knowledge from event logs, produced by information systems (Figure 1).

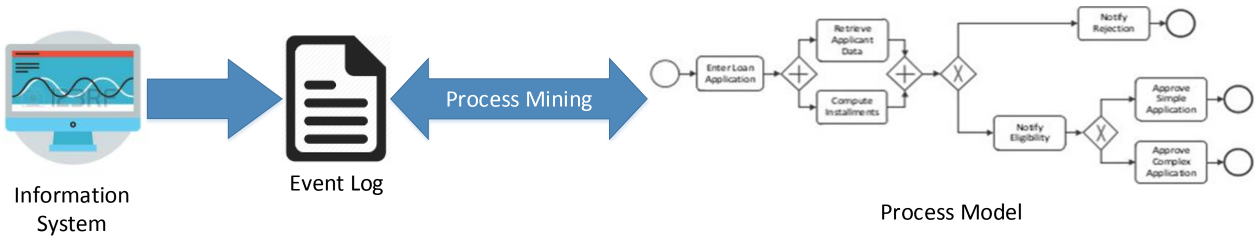


Figure 1. Process Mining overview

There are three types of process mining [2]: discovery, conformance, and enhancement. Discovery technique takes the event log as input and produces the model, as shown in Figure 1. The main goal of conformance is to compare a model with existing event log, which describes the same process, in order to check if reality conforms to the model and vice versa. Enhancement aims to extend or improve an existing process model using information about the actual process recorded in some event log [2]. In the thesis, we will focus on process discovery only.

A business process is a set of inter-related events, activities, and decisions, performed by multiple actors, which collectively allow an organization to deliver a product or service or achieve a business goal. An example of a business process is the process performed by a bank to approve loan applications, or the sales process of manufacturing company, which starts when a purchase order is received and ends when the corresponding invoice has been paid. This latter process is also called an order-to-cash process. An execution of a business process is called a case. For example, the concrete set of activities and decisions performed to fulfill a given purchase order constitutes one case of the order-to-cash process.

Every case consists of a sequence of events with the following attributes: timestamp, case identifier, and the name of the activity. By using timestamps, we can analyze the process from the time perspective, measuring the average execution time for the process overall or for each of the activities, estimating the waiting time between the activities, etc. Then, some process improvements can be done based on this information. Another purpose of timestamps is that they enable to order the events. However, it can be assumed that events arrive in a stream in chronological order, so the event sequence is already ordered. Therefore, to be able to discover control flow from event log every event should at least have the case identifier and activity name.

The quality of a process model discovered from an event log can be evaluated by four criteria [2; 3] (Figure 2):

- Fitness (measures the extent to which the traces in a log can be parsed by the discovered model [3]. A model with perfect fitness can replay all traces in the log)

- Precision (measure of additional behavior, which is not described in the log, allowed by a discovered model. A model with high precision can parse only the behavior specified in the event log)
- Generalization (measures how well the discovered model generalizes the behavior described in the log. For example, if a model can be discovered using 90% of the traces from the log and it can parse the rest 10% it has a good level of generalization)
- Simplicity (the model should be as simple as possible. In general, the complexity of a model can be defined as the total number of edges and nodes in the underlying graph. Therefore, a model with high simplicity has small amount of edges and nodes)

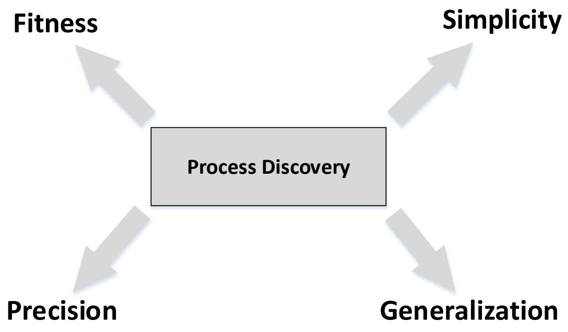


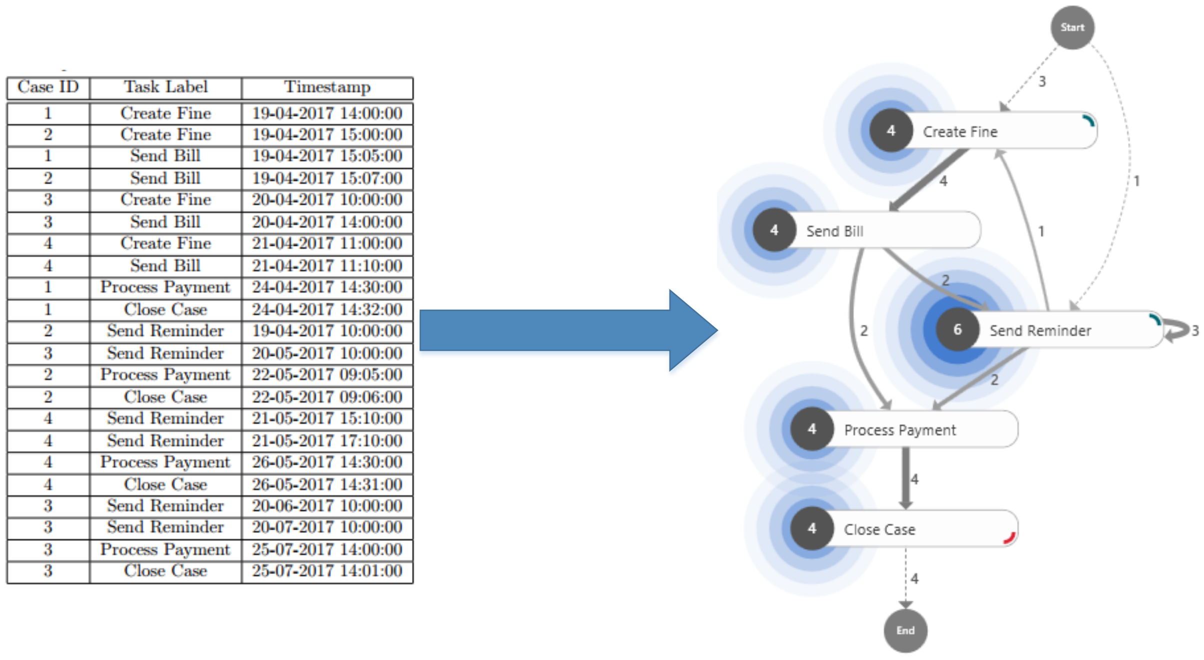
Figure 2. Four quality dimensions

2.2 Process Maps

A process map is directed weighted graph representation of a control-flow of the process, where:

1) nodes of the graph represent activities
2) edges represent causality relation
3) weights of edges represent the frequency of occurrence of the direct-follows relations (e.g. edge from activity A to activity B with weight 100 means that AB occurs 100 times)

The process map example is given in Figure 3:



Case ID	Task Label	Timestamp
1	Create Fine	19-04-2017 14:00:00
2	Create Fine	19-04-2017 15:00:00
1	Send Bill	19-04-2017 15:05:00
2	Send Bill	19-04-2017 15:07:00
3	Create Fine	20-04-2017 10:00:00
3	Send Bill	20-04-2017 14:00:00
4	Create Fine	21-04-2017 11:00:00
4	Send Bill	21-04-2017 11:10:00
1	Process Payment	24-04-2017 14:30:00
1	Close Case	24-04-2017 14:32:00
2	Send Reminder	19-04-2017 10:00:00
3	Send Reminder	20-05-2017 10:00:00
2	Process Payment	22-05-2017 09:05:00
2	Close Case	22-05-2017 09:06:00
4	Send Reminder	21-05-2017 15:10:00
4	Send Reminder	21-05-2017 17:10:00
4	Process Payment	26-05-2017 14:30:00
4	Close Case	26-05-2017 14:31:00
3	Send Reminder	20-06-2017 10:00:00
3	Send Reminder	20-07-2017 10:00:00
3	Process Payment	25-07-2017 14:00:00
3	Close Case	25-07-2017 14:01:00

Figure 3. Simple process map example

Unlike Petri net or BPMN model, it does not describe typical process design patterns, such as AND or XOR splits/joints, etc. Instead, it gives general knowledge about the process in an intuitive and understandable way, which is highly efficient in case of using in industry and business. Process maps help to identify bottlenecks, duplication, delays or gaps, to clarify process boundaries, process ownership, responsibilities and effectiveness measures or process metrics [4].

2.3 Data Streams

According to the definition, a data stream is an unbounded sequence of data items with a very high throughput [5]. Additionally, there are some assumptions, such as:

- Data have a small and fixed number of attributes
- The amount of data is infinite
- The memory is limited
- The time allowed to process an item is small

One type of data streams called event streams, which occur while working with business processes. The elements of event stream are event records, which have the same structure as events in the event log. In other words, event stream is the sequence of events before being recorded into an event log. Therefore, the main focus of this paper is done on event streams.

There are three types of data stream processing models [1]:

- ***Insert only model***
  Once an item is seen, it cannot be changed
- ***Insert-delete model***
  Items can be seen, deleted or updated
- ***Additive model***
  Each seen item refers to a numerical variable which is incremented

In the case of event streams, we are working with first type processing model because it does not make sense to delete events after they were observed.

3 Related Work

This chapter presents the state of the art of the problem. It gives an overview of process discovery techniques and challenges, data stream mining approaches and currently existing solutions for event streams process discovery. It also covers the process change problem and presents a possible way to deal with it. Finally, it describes cache replacement problem and cache replacement policies as a possible way to solve the event stream process discovery problem.

3.1 Research Method

Searching for relevant literature for selected topic is one of the most important stages of any research. The search was performed using following phrases:

- “event stream” AND “discovery” OR “event stream” AND “mining”
- “data stream” AND “discovery” OR “data stream” AND “mining”
- “process discovery”
- “process map”

Traditional search gives a high number of irrelevant results and on the other hand, it misses some relevant ones. Therefore, a more targeted search was completed based on papers that have been previously identified as relevant. Google Scholar was used to get the relevant papers and then the search of the papers that were cited in the relevant ones was performed.

The relevance of the paper can be estimated by following criteria:

- Is the paper about event streams or process maps discovery?
- Does the paper include a comparative analysis of the process discovery algorithms?
- Does the paper include real-life examples?
- Does the paper provide algorithm implementation details (e.g. detailed description of the algorithm, pseudocode examples)?

If the paper satisfies any of first two requirements it is considered to be relevant. Those that, in addition, include real-life examples and algorithm implementation details are highly preferred.

Before trying to implement the solution of the problem it was important to understand the current challenges and approaches. In order to identify process discovery challenges, the book [2] of W. M. P. Van der Aalst was used. In [6,7] and [6,8] the review of Heuristics and Fuzzy Miner respectively was performed. The research of M. M. Gaber and A. Zaslavsky [9] gives the overview of data stream mining techniques. G. Widmer and M. Kubat in their work [10] describe the problem of process change in time and introduce the possible ways how to deal with it. In the papers [1], [5] and [9] the review of currently existing event streams discovery approaches was done. Finally, in [12] we can find the principles of model update in a scalable and dynamic way.

3.2 Process discovery challenges

As it was specified earlier, Process Discovery faces many challenges. The biggest ones are following:

- *Noise*
  Noise is a term used to refer to rare (i.e. infrequent) behavior, which can be observed in the event log. The example of noise can be some exceptions in the execution of the system. The Pareto Rule can be applied to show the concept of the noise. It says that the 20% of the log may easily account for 80% of the variability in the process [2]. It

causes the difficulties in producing efficient models because the noise leads to over-crowding of the model since we focus not only on the actual behavior but also on the exceptional cases. This results in complicated, unreadable process models. Therefore, to be efficient, process discovery techniques should be able to deal with noise.

- *Incompleteness of event log*
  Another major concern for automated process discovery methods is incompleteness. It is not realistic to assume that log is complete and includes all possible behavior. Since event log is not complete, it is hard to build a good model based on it. Incompleteness leads either to excluding the missing behavior or to including unseen behavior, trying to fulfill the model. This can reduce the quality of the model. Therefore, the process model can be considered as just a view on reality [2].

- *Balance between underfitting and overfitting*
  There are two extreme cases of models, which can be created: the model, where everything is possible ("flower model" [2]) and the model, which allows only the behavior, specified in the event log ("enumerating model"). These models are examples of underfitting and overfitting respectively. It is crucial for process discovery techniques to find the balance between overfitting and underfitting. The final model should be general enough, but at the same time, it should not allow too much of behavior not observed in the event log.

- *Representational bias limitations*
  Any process discovery algorithm is effective within its representational bias – the class of models, which can be discovered. Therefore, in different cases, different algorithms should be used. All algorithms have their own representational limitations.

- *Support of different abstraction levels*
  Most of the algorithms assume that all events in the event log are on the same level of abstraction. Although, this simplify the discovery process, in real life actions belong to different abstractions levels. Therefore, to be more efficient on practice, process discovery algorithms should be able to deal with this problem.

3.3 Process discovery techniques

Considering the process maps specifics, the best process discovery algorithm, in this case, is Fuzzy Miner. In addition, in [7] A. Buratin and A. Sperduti developed the online version of Heuristics Miner for event streams process discovery, therefore we will present this algorithm as well.

Heuristics Miner algorithm takes into consideration the frequencies of the events and relations, thereby dealing with noise. By statistical measures and user defined thresholds, it discovers the dependencies from the event log and then builds a control-flow model on the basis of the observed directly-follows relations [5]. The dependency measure is calculated by frequency, e.g. the causal dependency between activities a and b is calculated as follows:

$$a \rightarrow b = \frac{|a>b|-|b>a|}{|a>b|+|b>a|+1} \in [-1,1] \quad (1)$$

Where $|a > b|$ is the frequency of occurrence of event b directly after the event a, i.e. the number of times that a directly-follows relation from a to b is observed in the log.
When the value is close to 1, there is a strong dependency between a and b and if the value is close to -1 there is a strong dependency between b and a. Using the dependency measure and a set of user-defined thresholds, Heuristics Miner builds a control-flow model in the form of Heuristics Net, directed graph in which nodes represent activities and edges

represent direct-follows dependencies. This graph is hereby called a Directly-Follow Graph (DFG). Moreover, outgoing/ingoing arcs of a node are annotated to XOR or AND splits/joins in case they constitutes exclusive choices or parallel alternatives respectively [5].
Given a log, the Heuristics Miner first calculates the dependency matrix wich contains the direct-follows relations observed between each pair of events. Next, based on the matrix, the algorithm constructs the dependency graph. The node corresponding to the activity for which none of the column/row entries in the matrix are positive is marked as start/end node. Finally, splits and joins types are identified using the AND measure. Given a dependency graph with activity *a* and two successor nodes *b* and *c*, their AND measure is following:

$$a \rightarrow (b \wedge c) = \frac{|b>c|+|c>b|}{|a>b|+|a>c|+1} \in [0,1] \quad (2)$$

When the value of AND measure is close to 1, the event *a* is an AND-split and events *b* and *c* can be executed in parallel. If the value of this measure is close to 0, then *a* is XOR-split and events *b* and *c* are in mutual exclusion.

Traditional process discovery algorithms have problems dealing with unstructured or semi-structured processes, producing complicated and unreadable models. These models are often „spaghetti-like“, showing all details without providing a suitable abstraction.

Fuzzy Miner algorithm is able to deal with unstructured processes, reducing complexity and improving comprehensibility [6]. It produces high-level view on the process, generalizes the process behavior when it is possible by removing of unimportant arcs in the directly-follows graph, clustering highly correlated nodes into a single node and removing isolated node clusters [8]. The result of the Fuzzy Miner discovery algorithm is process map – the general description of the control-flow of the process in a form of directed weighted graph.
For simplification of the process, Fuzzy Miner uses two metrics: significance and correlation. *Significance* measures the relative importance of behavior, either for individual events or for binary ordering relations. The simplest way to calculate the significance measure is by frequency, e.g. events with a higher frequency have more significance. *Correlation* metric shows how closely related two events following one another are. The measuring of correlation includes determining the overlap of data attributes associated with two events following one another or comparing the similarity of their event names. Highly significant behavior is preserved, less significant but highly correlated behavior is aggregated into clusters, while less significant and less correlated behavior is abstracted.
The first stage of the algorithm is to build the directly-follows graph. Then by significance and correlation metrics, it transforms the graph by edge filtering, aggregation, and abstraction. After this, it builds the model. The advantage of this algorithm is that depending on the specified constraints, such as minimum fuzzy support and minimum fuzzy confidence, it is able to provide the different views on the process with a different level of details. Because of its zoom-in and zoom-out capability, the fuzzy miner is a convenient method to deal with very large and highly unstructured event log. It is hence not surprising that commercial process mining tools such as Fluxicon Disco, Minit, myInvenio and Celonis relay mainly on variants of the fuzzy miner.

We note that both the heuristics miner and the fuzzy miner take as a starting point the directly-follows graph. The same is true of the alpha miner [2] and variants thereof, as well as the inductive miner [13]. In other words, the DFG is one of the fundamental primitives in the field of automated process discovery. Accordingly, in this thesis we focus on the problem of online automated discovery of DFGs, which we call herein "process maps", in line with the term in commercial tools.

3.4 Data stream mining approaches

In general, there are two data stream mining approaches: data-based and task-based [9].

The main concept of **data-based** solutions is to analyze the only fragment of a stream or to summarize the data. The examples of data-based techniques are sampling, aggregation, load shedding and sketching.
The idea of *sampling* technique is that by studying examples from a data stream, it is possible to generalize the results back to the whole data stream. It is a quite old technique that has been used for a long time. The problem with using sampling when working with data streams is the unknown dataset size. Therefore, it is a question how much examples are needed to achieve relevant results. Moreover, some of the items that are ignored can be interesting and meaningful.
The *load shedding* technique aims to drop a sequence of data streams, decreasing the amount of data, that need to be processed. However, it is difficult to use with mining algorithms because it drops chunks of data streams that can be used in the structuring of the generated models or it might represent a pattern of interest in time series analysis [9].
According to [9], *sketching* is the process of vertically sampling the incoming stream, with a random projection of a subset of features. The disadvantage of sketching is problems with accuracy.
The main goal of *aggregation* is to summarize the incoming stream using statistical measures such as mean, median, etc. This aggregated data then can be used as input for mining algorithms, but it has problems dealing with data with highly fluctuating distributions.

Task-based approaches designed to address the computational challenges of data stream processing. The solutions of this type are represented by approximation algorithms, sliding window, and algorithm output granularity.
The *approximation algorithms* aim to extract approximate solution, with a possibility to define error bounds on the procedure.
The main idea of *sliding window* is to concentrate only on “fresh” events. The analysis is performed giving more importance to recent data, considering only summarization of the old ones.
The *algorithm data granularity* is resource-aware data analysis approach that deals with fluctuating high data rates according to the available memory and processing speed [9].

In addition, the following data mining and machine learning techniques can be applied to analyze data streams: clustering, classification, frequency counting, change diagnosis and time series analysis. Taking into consideration the characteristics of process maps, the main focus should be done on frequency counting.

3.5 Process change

As it was mentioned earlier, processes can change in time. There are following types of changes that can be observed: the *drift* of the process model, the *shift* of the process model and cases *distribution change* [9]. The drift of the model refers to a gradual change of the underlying process. Model shifts happen when a change between two process models is sharper. Although the original process can stay the same during the time, the cases distribution may change, therefore causing changes. For example, in a production process of a company selling clothing, the items involved in incoming orders during winter will follow a

completely different distribution than in the summer. Such distribution change may significantly affect the relevance of specific paths in the control-flow of the involved process.
The general approach to deal with concept drift has three main characteristics [10]: 1) keeping only a window of currently trusted examples and hypothesis; 2) storing concept descriptions and re-using them if the previous context re-appears; 3) controlling both of these functions by the heuristic that constantly monitors the system's behavior.
Effective process mining algorithm has to be able to detect context changes without being explicitly informed about them and quickly adapt the model accordingly to these changes.
One possible approach is to trust only the most recent examples since the processes tend to change in time. The set of these examples called a "window". Newly arrived examples are added to the window and the oldest ones have to be deleted from it. The model has to allow the behavior from the window. In the most simple case, the window has fixed size and the oldest example will be deleted when a new one is observed. This approach is called "time-based forgetting" [10].

3.6 Online process discovery

The general idea of event streams process discovery is described in Figure 4:

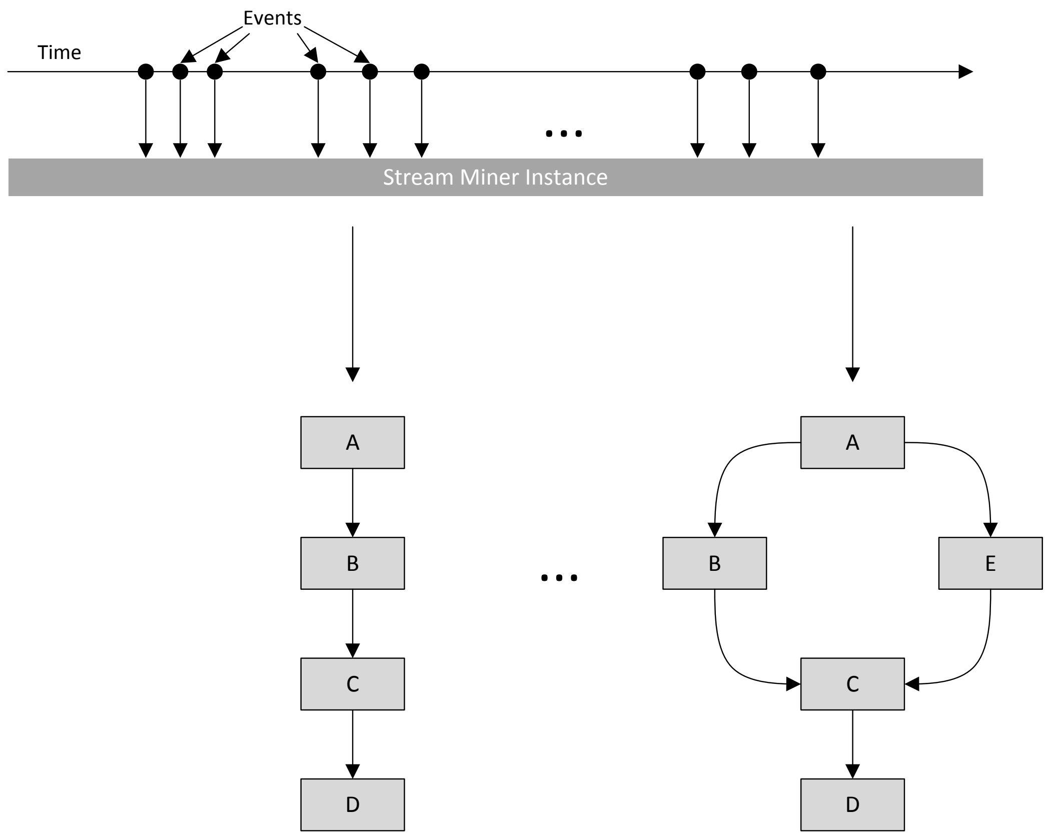


Figure 4. General idea of event stream process mining

Currently, there are available several algorithms to deal with data streams such as Sliding Window, Sticky Sampling, Lossy Counting and Lossy Counting with Budget.

The main idea of **Sliding Window** algorithm **(**Figure 5**)** is to collect events for a certain span of time and then apply a standard offline process discovery approach. This approach has several advantages such as the possibility to mine the log using any process mining algorithm. However, the notion of history is not very accurate: only the more recent events are considered, and equal importance is assigned to all of them. In addition, the model is not constantly updated since each newly received events triggers only the update of the memory and not necessarily an update of the model. Model updating with every event occurrence will cause computational problems. The time required by this approach is completely unbalanced. When the new event arrives, inexpensive operations are performed (shift and insert), instead of when the model needs to be updated, the log, retained in memory is mined from scratch. Therefore, each event is handled at least twice: the first time to store it into a log and subsequently any time the mining phase takes place on it. In mining event streams, it is more desirable to process each event only once.

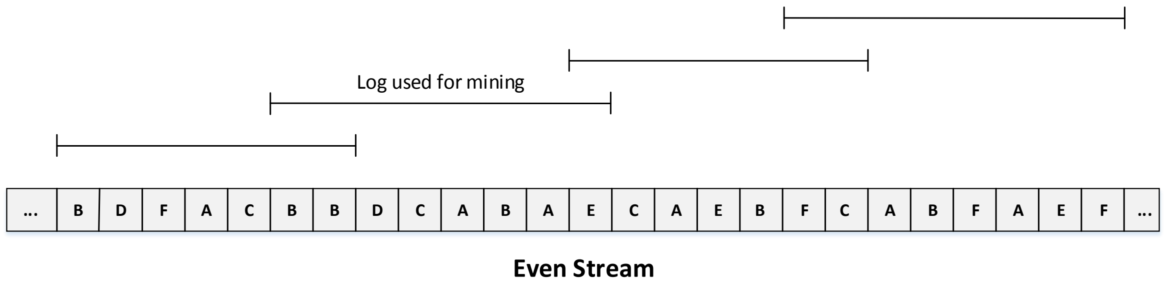


Figure 5. Sliding Window algorithm

Sticky Sampling (SS) (Figure 6) is a probabilistic algorithm, an adaptation of an existing technique, used for approximate frequency count. It fails to provide correct answers with a miniscule probability of failure [11]. The algorithm takes three parameters such as support S, error ϵ, and probability of failure δ. S is a tuple of the form (e, f), where f estimates the frequency of an element e in the stream. Initially, it is empty and the sampling rate r is set to 1. The probability of selection of the element is $\frac{1}{r}$. For each incoming element e, if an entry for e already exists in S, the corresponding frequency has to be incremented. In another case, we sample the element with rate r. If this element is selected by sampling, we add an entry $(e, 1)$ to S [11]. Otherwise, we ignore the element and move on to the next one in the stream. Sampling rate r increases logarithmically proportional to the size of the stream [11]. Whenever the sampling rate r changes, for each entry (e, f) we will toss an unbiased coin and in the case toss is not successful, the frequency f of the element e will be decreased by 1. When frequency will reach 0, the corresponding entry will be deleted. The space complexity of the algorithm is independent of the current length of the stream.

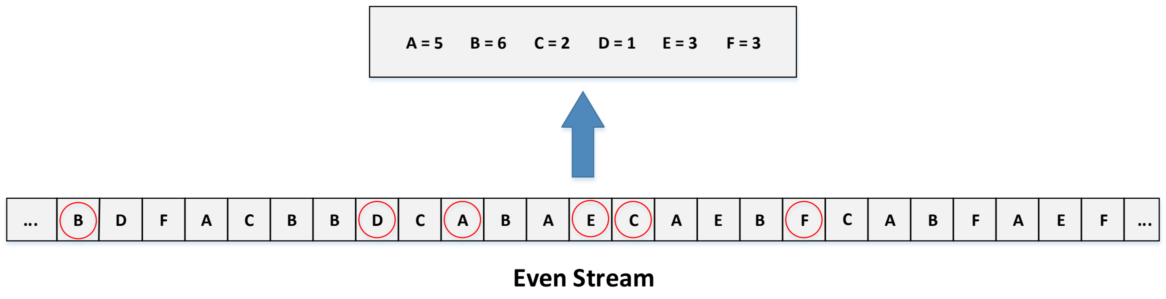

A = 5 B = 6 C = 2 D = 1 E = 3 F = 3
... B D F A C B B D C A B A E C A E B F C A B F A E F ...
Even Stream

Figure 6. Sticky Sampling algorithm

Lossy Counting (LC) (Figure 7) algorithm is an alternative to Sticky Sampling, but it is a deterministic approach. The main idea is to consider aggregated representations of the latest observations instead of storing repetitions of the same event, in order to save space, they store a counter keeping trace of the number of instances of the same observation. The stream is divided into buckets of width $w = \left\lceil \frac{1}{\epsilon} \right\rceil$, where $\epsilon \in (0,1)$ represents the maximum approximation error allowed. The current bucket is identified with $b_{cur} = \left\lceil \frac{N}{w} \right\rceil$, where N is the progressive events counter. The basic data structure used by Lossy Counting is a set of tuples of form (e, f, Δ) where: e is an element of the stream, f is the estimated frequency of item e and Δ is the maximum number of times event e already occurred. Every time a new event is observed, the algorithm looks whether the data structure contains an entry for the corresponding element. If such entry exists then its frequency value is incremented by one, otherwise, a new tuple is added: $(e, 1, b_{cur} - 1)$. Every time $N \, mod \, w = 0$, the algorithm cleans the data structure by removing the entries that satisfy the following inequality: $f + \Delta \leq b_{cur}$. Such inequality ensures that every time the cleanup procedure is executed, $b_{cur} \leq N$. The main drawback of this algorithm is that the only parameter of Lossy Counting is the maximum allowed error in the frequency approximation and no information on space usage can be provided. Lossy Counting drops low frequent items fast and only elements with high frequency survive. Therefore, it can cope with noise problem of process discovery. Sticky Sampling performs worse because of its tendency to remember every unique element that gets sampled [11].

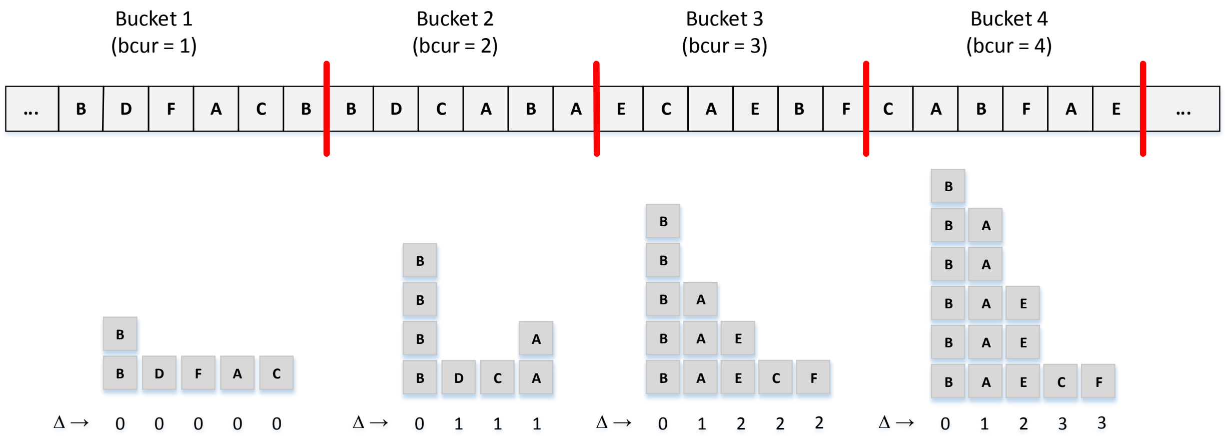


Figure 7. Lossy Counting Algorithm

Lossy Counting with Budget (LCB) is the modification of Lossy Counting algorithm. It uses the same data structure to store the incoming events, but, unlike Lossy Counting, it allows to control the maximum space used to store these data structures. This algorithm divides the stream into buckets of variable sizes. Each bucket will contain all occurrences that can be accommodated into the available budget. The approximation error for a given bucket B_i will depend on the bucket size and is defined as $\epsilon_i = \frac{1}{B_i}$. Buckets of variable sizes imply that the cleanup operations do not occur always at the same frequency, but only when the current bucket is not able to store the newly observed event. The remove condition is the same as in Lossy Counting. The current bucket, however, is updated only when no more items can be stored. Moreover, the deletion condition may not be satisfied by any item, thus not creating space for the new observation. In this case, the algorithm has to increase the current bucket size arbitrarily, until at least one item is removed. The basic structure of the procedure has several differences comparing to Lossy Counting. The cleanup operations must be performed before inserting an item (not after as in Lossy Counting) and the cleanup has to update the current bucket accordingly. Currently Lossy Counting with Budget it the most efficient technique

3.7 Model update

In order to update the model in an efficient way, the online discovery algorithm should be divided into two separate parts: run-time footprint calculation and scheduled footprint interpretation. The footprint in this context is the abstract representation of the entire process taken in a certain time. Footprint consists of the set of activities and relations together with information about their frequency. The main responsibility of run-time footprint calculation component is to update the footprint when a new event occurs. Taking into consideration the high frequency of events occurrence, the algorithmic execution time needs to be minimized. Since the process tends to change in time, the old events should have a small influence on the process model. The footprint has to be dynamically updated, giving more importance to the recent events, and using the summarization of the old ones. In order to meet the scalability requirement, the footprint update should take a fixed amount of time, independent of the total number of previously occurred events and traces [12]. To deal with noise, the events that have not been seen for a long time should be deleted. In the schedule footprint interpretation component, the model is discovered from the current footprint in a scheduled, reoccurring way. Therefore, it has less execution time constraints. The model can be updated after some fixed amounts of traces occurred or after every *n* minutes.

3.8 Cache Replacement Problem

As it was mentioned earlier, it is not possible to keep the whole event stream in memory. Therefore, we need a mechanism to “free-up” space once a certain limit is reached. The problem of clearing the memory for upcoming data can be seen as a cache replacement problem.

Nowadays, the web is the most important source of information and communication in the world. The majority of web objects are static, therefore caching them at HTTP proxies can reduce network traffic and response time [14]. However, since the size of the cache is limited, there have to be wise strategies that identify which objects have to stored in the cache and which have to be thrown away in order to clear the space for new ones. These strategies are called *cache replacement policies*.

Web objects have many characteristics. For the purpose of caching, recency, frequency and size are the most commonly used properties. Accordingly, the cache replacement policies can be classified into three categories: recency oriented, frequency oriented and size oriented policies. In some cases, policies of one type can use methods from another type in order to choose the objects for removal (this can happen when the main characteristics of objects are equal).

Recency-Based Strategies

This set of strategies is derived from a property known as temporal locality, the measure of how likely an object is to appear again in a request stream after being requested within a time span [15]. These strategies use recency information as a significant part of their victim selection process. Recency based strategies are typically straight forward to implement taking advantage of queues and linked lists.

1) *Least Recently Used (LRU)*
   One of the most known and commonly used strategies. Replaces the block in the cache that has not been used for the longest period of time. From the basics of temporal locality, the blocks that have been referenced in recent past will likely be referenced in the near future. This policy works well when there is a high temporal locality of references in the workload.

2) *LRU-Threshold*
The modification of LRU policy. An object is not permitted into the cache when its size exceeds given threshold.

3) *Pitkow/Reckers Strategy*
Objects that are most recently referenced within the same day are differentiated by size, choosing the largest first. Objects referenced not in the same period are sorted by LRU.

4) *Value Aging*
Defines a characteristic function based on the time of a new request to object and removes an object with the smallest value of the function. The function is the following:

$$R_i = C_i * \sqrt{\frac{C_i - T_i}{2}} \quad (3)$$

where C_i is the current time and Ti – last time the object was referenced.

5) *LRU**
Combines an LRU list and a request counter. When an object enters the cache, its request counter is set to 1 and it is added to the front of the list. On a cache hit, its request counter is incremented by 1 and also moved to the front of the list. During victim selection, the request counter of the least recently used object (the tail of the list) is checked. If it is zero, the object is removed from the list; if it is not zero, its request counter is decremented by 1 and moved to the front of the list.

6) *Segmented LRU*
Divides the cache into two parts. The first part is known as the unprotected segment and the second - the protected segment. The strategy requires space set aside for the protected segment, known as A. Objects that belong to this segment cannot be removed from the cache once added. Both segments are managed by LRU replacement strategy. When an object is added to the cache, it is added to the unprotected segment, removing only objects from the unprotected space to make room for it. There is an implicit size threshold for objects, where the minimum object size allowed to be cached is min(A, M – A). Upon a cache hit of an object, it is moved to the front of the protected segment. If the object is in the unprotected segment and there is not enough space in the protected segment, the LRU strategy is applied to the protected segment, moving objects into the unprotected segment. [15]

7) *Most Recently Used (MRU)*
In contrast to LRU, it removes the most recent objects.

Frequency-Based Strategies

Frequency based strategies use a property of request streams known as spatial locality, the likelihood that an object will appear again based on how often it's been seen before [15].

1) *Least Frequently Used (LFU)*
Removes the object with the smallest frequency of occurrence.

2) *LFU-Aging*

The LFU policy can suffer from cache pollution (an effect of temporal locality): if a formerly popular object becomes unpopular, it will remain in the cache for a long time, preventing other newly or slightly less popular objects from replacing it. This strategy tries to solve this problem by introducing an aging factor. When the average of all the frequency counters in the cache exceeds a given average frequency threshold, then all frequency counts are divided by 2. There is also a maximum threshold set that no frequency counter is allowed to exceed.

3) *LFU with Dynamic Aging (LFU-DA)*
A parameterless policy which is less complex and easier to manage. For each object, the key value K_i is calculated such as:

$$K_i = F_i + L \quad (4)$$

where F_i is the frequency of object i and L is the dynamic aging factor. Initially, it is set to 0, but with a removal of object i, its value is set to K_i.

4) *α-Aging*
Periodic aging method, which can use varying periods and a range [0,1] for its aging factor α. Each object uses a value K which is equal to the frequency of the object. At the end of every period an aging factor is applied to each object:

$$K_{new} = a * K, 0 \leq a \leq 1 \qquad (5)$$

The object with the smallest value of K will be removed.

5) *HYPER-G*
First, removes the least frequent objects. Then if the objects have the same frequency, the LRU is applied. Finally, if it is not enough to choose one object for deletion, the largest object will be chosen.

Size-Based Strategies

These strategies aim to minimize the memory consumption, by removal of the largest objects.

1) *SIZE*
This strategy removes the largest objects first, to minimize the amount of memory used. In the case when objects have the same size, LRU is applied to choose the object for deletion.

2) *LOG2-SIZE*
Sort objects by their floor[log(size)], differentiating objects with the same value by LRU. This strategy tends to apply LRU more often comparing to SIZE strategy.

3) *Greedy-Dual Size (GDS)*
This strategy replaces the object with the smallest key value for a certain utility (cost) function. When an object i is requested its key value is calculated as follows:

$$K_i = \frac{C_i}{S_i} + L \quad (6)$$

where C_i is the cost associated with bringing object i into the cache, S_i – size of object and L is a running age factor that starts at 0 and is updated for each replaced object j to the priority key of this object: $L = K_j$. In the case when C = 0, the removal procedure depends only on the size of objects.

4) *Greedy-Dual Size with Frequency (GDSF)*
The modification of Greedy-Dual Size strategy. Take into account the frequencies of objects. The key values will be calculated in the following way:

$$K_i = F_i * \frac{C_i}{S_i} + L \quad (7)$$

In the case of process discovery, all events have equal size, and hence size replacement policies are not very relevant when it comes to determine which events should be kept and which ones can be replaced. Accordingly, in this thesis we will consider recency and frequency-based policies.

3.9 Summary

Currently, all of the presented approaches for online automated process discovery are based on Heuristics Miner algorithm and designed for discovery of BPMN process models. However, as it was mentioned earlier, in order to be able to cope with unstructured processes, we will focus on the online discovery of process maps. The approach to be followed is to adapt the Fuzzy Miner algorithm by using a combination of Lossy Counting with Budget technique together with cache replacement policies in order to enable real-time discovery. We retain from the overview provided here that the LRU and LFU policies are among the most relevance cache replacement policies to manage the requirement of a fixed-size budget for storing events of completed traces. In addition, we will consider the LFU-DA policy in order to cope with concept drifts.

4 Contribution

In the following section, the solution will be presented in the form of the process map discovery algorithm, which is able to deal with event streams. It presents the algorithm including pseudocode and architecture of the solution.

4.1 Requirements to solution

As it was discussed earlier the following requirements to discovery algorithm were identified:

- The algorithm has to minimize the amount of memory used
- The algorithm execution time need to be minimized
- The update of the current state of the process has to be done after observing every new element in event stream
- The model update has to meet scalability requirements
- Algorithm has to produce the actual current state of the process, dealing with concept drifts
- Algorithm has to deal with noise
- The result of the execution of the algorithm is a process map

The most efficient algorithm to produce process maps is Fuzzy Miner. It uses two major metrics to build the model: significance and correlation. Significance can be calculated by frequencies of activities and direct-follows relations. In order to estimate the correlation, we have to know the performers of the activities, data they are working with, etc. In most of the cases, an incoming event has a structure (c, a, t), where c represents case identifier, a is an activity which is done and t is a timestamp. Therefore, we will focus on significance metric only. In this case, the produced model can be filtered by the minimal support of activity or path.

In section 3.6. currently existing frequency counting approaches were presented. Sliding Window algorithm takes events for a certain period of time, produces temporal event log and then apply an offline version of the process discovery algorithm. This approach does not update the model constantly since each new event triggers only the update of the memory but not necessarily the model. In addition, the time required by this approach is completely unbalanced, and the algorithm performs a lot of inexpensive operations. Therefore, it does not meet the requirements specified above. Lossy Counting and Sticky sampling are two alternative approaches, but based on the results of analysis in [11], Lossy Counting performs better. This algorithm deals with noise removing low frequent behavior. In addition, it constantly updates the abstract footprint of the model with minimum time. One disadvantage of this algorithm is that it does not take into consideration the memory limitation. The Lossy Counting with Budget is a modification of Lossy Counting algorithm, which deals with this problem. Based on retrieved relations and their frequencies the control-flow of the process can be built.

However, the fact that appearance of a new case can trigger the deletion of event or relation, therefore unnecessary losing some information does not satisfy the specified requirements. Therefore, this algorithm has to be rewritten. Cache replacement policies will take responsibility of objects deletion. We will take three of them (LFU, LRU, LFU-DA) and compare the results.

The model update will be done in scheduled way after the occurrence of n events or after n seconds/minutes. In order to identify the criteria for scheduling, we will need to consider the frequency of occurrence of a new event. On the one hand, in large enterprises, where information systems produce a huge amount of data with high frequency, the model update has to be done after every *n* second. On the other hand, in the case of low frequency of new event occurrence, the model can be updated after every *n* received events. Moreover, the update of the model can even be done once a day in case it used for monitoring purposes. Therefore, the scheduling type strongly depends on the business case. The update can be done automatically or manually.

4.2 Architecture of the solution

The high-level architecture of the solution is described by Figure 8:

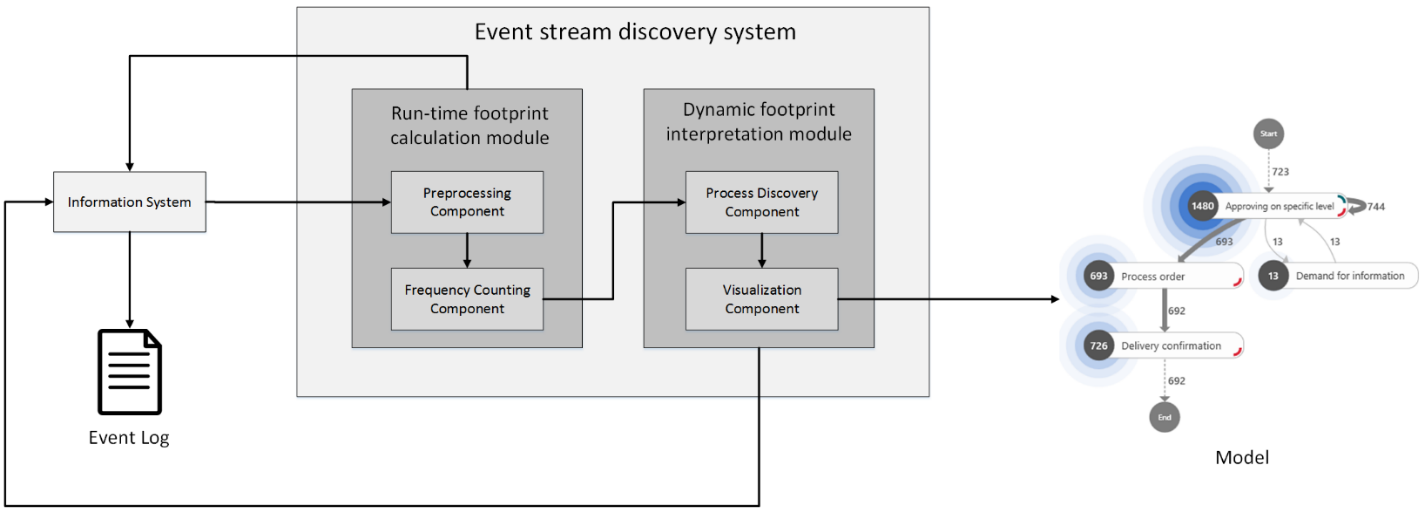

Event stream discovery system
Run-time footprint calculation module
Dynamic footprint interpretation module
Information System
Preprocessing Component
Frequency Counting Component
Process Discovery Component
Visualization Component
Event Log
Start
723
1480
Approving on specific level
744
693
13
13
693
Process order
13
Demand for information
692
726
Delivery confirmation
692
End
Model

Figure 8. The architecture of solution

It consists of two main modules which are working in parallel but with different frequencies. The first one updates the footprint of the current state of the process with an occurrence of every event, while the second updates the dependency graph and visualize the current model in scheduled recurring way.

According to this chart, there are four main components:

1) *Preprocessing component*

This component is responsible for extraction of event attributes. In an offline scenario, every event is represented by a row in Event Log table (CSV file for example). In online scenario event which is recorded by the information system is a string of the form of:

"case_id<***separator***>activity<***separator***>timestamp",

where separator is a character delimiter such as comma (','), semicolon (';') or dot ('.') However, the attributes can have a different order (e.g. "1, Sent to Customer, 02-10-16 17:40" and "02-10-16, 1, Sent to Customer") or different separators. In addition, some attributes can be empty or even miss. The main problem is related to timestamps. There many notations of timestamps (e.g. "07.11.2016 17:40" and "02-10-16 17:40") and sometimes they are not recorded by information systems. A lot of discovery techniques rely on timestamps, but, in the case of working with event streams, to discover control-flow of the process case identifier and activity name are only needed. Therefore, many data quality problems can be eliminated. In order to do further analysis, find bottlenecks and other possible problems we need timestamps. The main idea of this component is to use regular expressions to extract the necessary attributes values and to create an object of the class Event based on it (Figure 9):

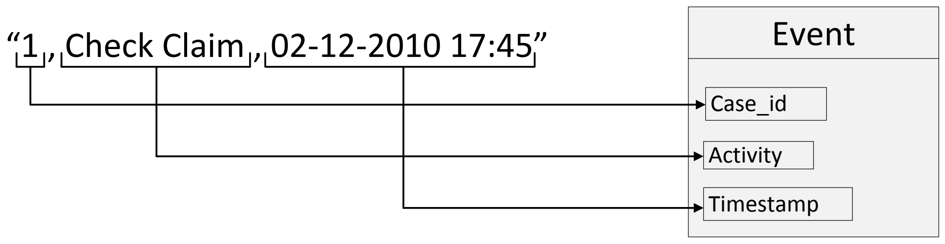


Figure 9. Attributes extraction

2) *Frequency Counting Component*

This component takes an important part in the solution because it has to work online, taking into consideration time and memory limitations. Time needed to process an event has to be minimized as well as used memory. As it was discussed earlier the Lossy Counting with Budget frequency algorithm is chosen but it has to be modified. Let us put all changes into the list:

- The budget will specify the maximum amount of activities and relations that can be stored in memory. Since activities and relations represent nodes and edges of the graph model, budget now will have more practical interpretation
- Deletion procedure will be triggered only when the total amount of activities and relations exceeds the budget (appearance of new case will not trigger deletion of activity or relation anymore)
- All running cases will be kept in memory and finished cases will be deleted. In order to understand whether the case is finished, the notion of activity type is introduced.
- Cache replacement policies will be responsible for deletion of objects
- The concept of buckets is eliminated. Instead of it the last seen index metric will be used for every object to define its „age“
- There is no need for case object to store more attributes than case_id and last activity

In order to apply modified Lossy Counting with Budget method to Fuzzy Miner algorithm we need to define following variables:

e – event, tuple (*c*, *a*, *t*, *type*), where *c* – case_id, *a* – activity, *t* – timestamp, *type* – type of event
ES – event stream
B – available budget (used in order to specify memory constraints)
S_a – the set of activities, each element of the set is tuple (*a*, *f*, *idx*), where *a* – activity, *f* – estimated frequency, *idx* - last seen index
S_c – the set of cases, each element is a tuple (*c*, *a*), where *c* – case_id, *a* – the last activity in the case
S_r – the set of direct succession relations, each element is a tuple of (*r*, *f*, *idx*), where *r* – relation (start_activity, end_activity), *f* – frequency, *idx* – last seen index

Structures S_a and S_r store the frequency of activities and relations respectively. Additional structure S_c is used to hold the running cases. Here is the last activity of case has to be updated every new event of the current case arrives. Every time a new event e = (c_i, a_i, t_i) is observed, S_a is updated with the new observation of the activity *a*. After that, the procedure checks if there is an entry in S_c associated to the case id of the current event. If not, a new entry is added to S_c. Otherwise, the case is already running and the observed event is not the start event. Therefore we are observing the direct-follows relation. The algorithm builds the relation r (last activity of the case, a_i) and updates its corresponding frequency in S_r. Finally, if the case c is finished, it will be removed from S_c. S_a and S_r will share the memory, thus the appearance of new activity can trigger the deletion of relation. When an entry is removed from S_c, the corresponding activity can not appear in any relations in S_r.

Three well-known cache replacement policies such as LFU, LRU, and LFU-DA will be used as the deletion mechanisms. Therefore, we will have three modifications of Lossy Counting with Budget algorithm:

1) LCB with LRU policy

Here the pointer in the stream will be continuously updated with an occurrence of every event. Objects of S_a and S_r will have attribute last seen index, which will tell about the recency of an element. When the total amount of activities and relations will exceed the budget, the LRU policy will be triggered. When the least recent object is relation it will be deleted from S_r. However, when the least recent object is activity, in addition to removal of this activity, all relations associated to activity have to be deleted as well.

Algorithm 1: LCB with LRU policy

```
Input: ES: event stream; B: available budget
S_a ← null          /* Initially empty set */
S_c ← null          /* Initially empty set */
S_r ← null          /* Initially empty set */
idx ← 1       /* Initial pointer in the stream */

1  forever do
2     e ← observe(ES)
3     if analyse(e) then
4          if activity $a$ is already in the S_a then
5               Increment the frequency of $a$ in S_a
6               Update lastSeenIndex of $a$ in S_a
7          else
8               if $|S_a| + |S_r| = B$ then
9                    Apply deletion procedure
10              end
11              Insert $(a, 1, idx)$ in S_a
12         end
13         if case $c$ is already in the S_c then
14              if relation r(a, a_i) exists in the S_r then
15                   Increment the frequency of r in the S_r
16                   Update lastSeenIndex of r in S_r
17              else
18                   if $|S_a| + |S_r| = B$ then
19                        Apply deletion procedure
20                   end
21                   Insert $(r, 1, idx)$ in S_r
22              end
23              update $a$ in $(c, a)$
24              if case c is finished then
25                   Delete c from S_c
26              end
```

```
27          else
28              Insert $(c, a)$ in $S_c$
29          end
30          idx ← idx + 1
31      end
32  end

Deletion Procedure:
Input: $S_a$, $S_c$, $S_r$, B

1   $min_a$ ← min($idx$ in $(a, f, idx) \in S_a$)
2   $min_r$ ← min($idx$ in $(r, f, idx) \in S_r$)
3   if ($min_a > min_r$) then
4       Remove $(r, f, idx)$ from $S_r$ where $idx = min_r$
5   else
6       Remove $(a, f, idx)$ from $S_a$ where $idx = min_a$
7       for each $(r, f, idx) \in S_r$ where $start\ or\ end\ activity = a$ do
8           Remove $(r, f, idx)$ from $S_r$
9       end
10  end
```

2) **LCB with LFU policy**

In contrast to LRU, there is no need to keep the pointer on the event stream and store the last seen index for every object. Therefore, the size of objects can be shortened and the memory consumption will decrease. The deletion procedure will be similar to the first case, but we will drop the object with the smallest frequency.

Algorithm 2: LCB with LFU policy

```
Input: ES: event stream; B: available budget

$S_a$ ← null        /* Initially empty set */
$S_c$ ← null        /* Initially empty set */
$S_r$ ← null        /* Initially empty set */

1   forever do
2       e ← observe(ES)
3       if analyse(e) then
4           if activity $a$ is already in the $S_a$ then
5               Increment the frequency of $a$ in $S_a$
6           else
7               if $|S_a| + |S_r| = B$ then
8                   Apply deletion procedure
9               end
10              Insert $(a, 1)$ in $S_a$
11          end
12          if case $c$ is already in the $S_c$ then
13              if relation (a, $a_i$) exists in the $S_r$ then
14                  Increment the frequency of r in the $S_r$
15              else
16                  if $|S_a| + |S_r| = B$ then
17                      Apply deletion procedure
18                  end
```

```
19                  Insert $(r, 1)$ in $S_r$
20              end
21              update $a$ in $(c, a)$
22              if case c is finished then
23                  Delete c from $S_c$
24              end
25          else
26              Insert $(c, a)$ in $S_c$
27          end
28      end
29  end
```

Deletion Procedure:
Input: S_a, S_c, S_r, B

```
 1  $min_a \leftarrow$ min(f in $(a, f) \in S_a$)
 2  $min_r \leftarrow$ min(f in $(r, f) \in S_r$)
 3  if ($min_a > min_r$) then
 4      Remove $(r, f)$ from $S_r$ where f = $min_r$
 5  else
 6      Remove $(a, f)$ from $S_a$ where f = $min_a$
 7      for each $(r, f) \in S_r$ where $start\ or\ end\ activity = a$ do
 8          Remove $(r, f)$ from $S_r$
 9      end
10  end
```

3) LCB with LFU-DA policy

This approach will be the closest to original Lossy Counting with Budget method because it will consider frequency and age of elements (unlike first two approaches where only one metric was considered, either frequency or age). As it was stated in section 3.8, LFU-DA policy uses a calculation of key value K in order to drop the elements. K is calculated as $K_i = F_i + L$, where F_i is frequency and L is a dynamic aging factor. Initially, L is set to 0, thus before the first deletion, we are calculating just frequencies as in the second approach (LFU). However, after deletion of object j, L is updated and set to K_j.

Algorithm 3: LCB with LFU-DA policy

Input: ES: event stream; B: available budget

```
$S_a \leftarrow$ null        /* Initially empty set */
$S_c \leftarrow$ null        /* Initially empty set */
$S_r \leftarrow$ null        /* Initially empty set */
$L \leftarrow 0$          /* Initial dynamic aging factor */

1  forever do
2      e ← observe(ES)
3      if analyse(e) then
4          if activity $a$ is already in the $S_a$ then
5              Increment the frequency of $a$ in $S_a$
6          else
7              if $|S_a| + |S_r| = B$ then
8                  Apply deletion procedure
```

```
9                 Update L
10            end
11            Insert $(a, 1, L)$ in $S_a$
12        end
13        if case $c$ is already in the $S_c$ then
14            if relation $(a, a_i)$ exists in the $S_r$ then
15                Increment the frequency of r in the $S_r$
16            else
17                if $|S_a| + |S_r| = B$ then
18                    Apply deletion procedure
19                    Update L
20                end
21                Insert $(r, 1, L)$ in $S_r$
22            end
23            update $a$ in $(c, a)$
24            if case c is finished then
25                Delete c from $S_c$
26            end
27        else
28            Insert $(c, a)$ in $S_c$
29        end
30    end
31 end
```

Deletion Procedure:
Input: S_a, S_c, S_r, B

```
1  $min_a \leftarrow min(f + \Delta \text{ in } (a, f, \Delta) \in S_a)$
2  $min_r \leftarrow min(f + \Delta \text{ in } (r, f, \Delta) \in S_r)$
3  if ($min_a > min_r$) then
4      Remove $(r, f, \Delta)$ from $S_r$ where $f + \Delta = min_r$
5  else
6      Remove $(a, f, \Delta)$ from $S_a$ where $f + \Delta = min_a$
7      for each $(r, f, \Delta) \in S_r$ where $start\ or\ end\ activity = a$ do
8          Remove $(r, f, \Delta)$ from $S_r$
9      end
10 end
```

The execution of this module results in the list of frequencies for activities and direct-follows relations. Based on this data it is possible to identify the control-flow structure of the process. The proposed approaches allow us to control the loss. Since the model creation now depends only on the activities and relations, it is possible to calculate which budget is needed to get the lossless model by using the following formula:

$$B = \frac{N(N+1)}{2} + N \quad (8)$$

where N is the total number of distinct activities.

Let us take as an example Cloud Order Human Anonymized log, provided by company Minit. The model of the process described in the log (Figure 10):

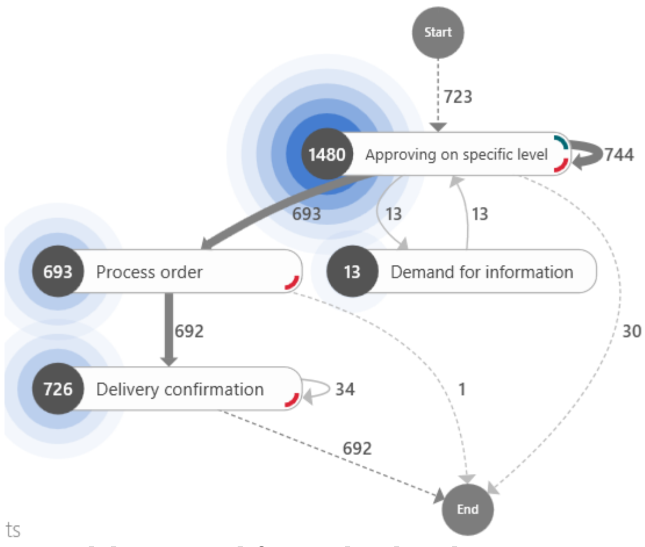


Figure 10. Model extracted from Cloud Order Human Anonymized log

The budget to get lossless model is the following:

$$B = \frac{4 * 5}{2} + 4 = 14$$

The model obtained with this budget is described by Figure 11:

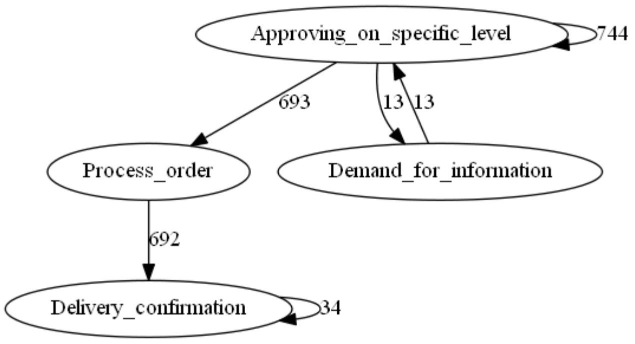


Figure 11. Model discovered by the system

It can be easily seen that the obtained model is equal to the original.

3) *Process Discovery Component*

It is an intermediate component between frequency counter and visualization component that applies process mining techniques in order to discover the control-flow of the process, to identify parallelism, loops, etc. In the case of process map it is impossible to say whether it is parallelism or choice (there is no such notion as gate), thus both of them can be discovered in the same way: if there are exist relations (a,b) and (b,a) in the same time then a and b are parallel. In the case of discovering BPMN model, the technique, mentioned in Heuristics Miner can be applied (*see Section 3.3*).

4) *Visualisation Component*

Based on the results of process discovery it builds the graphical view of the process.

4.3 Implementation

The solution is built using Cortana Intelligence. It is a cloud-based solution for work with big data, which provides advanced analytics tools (Figure 12).

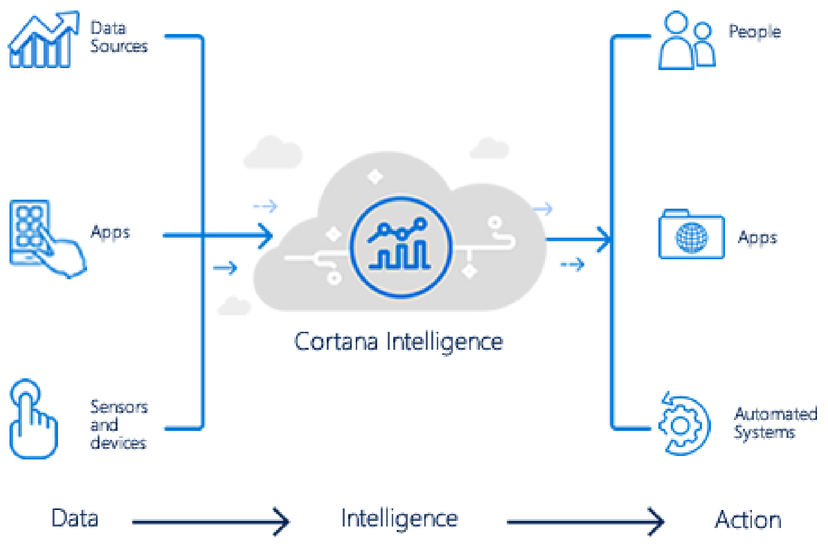


Figure 12. Role of Cortana Intelligence (Source: http://bit.ly/1RTOHRg)

Cortana Intelligence Suite includes Machine Learning and Analytics tools as well as Information Management and Visualization solutions. The full stack of tools is given in Figure 13:

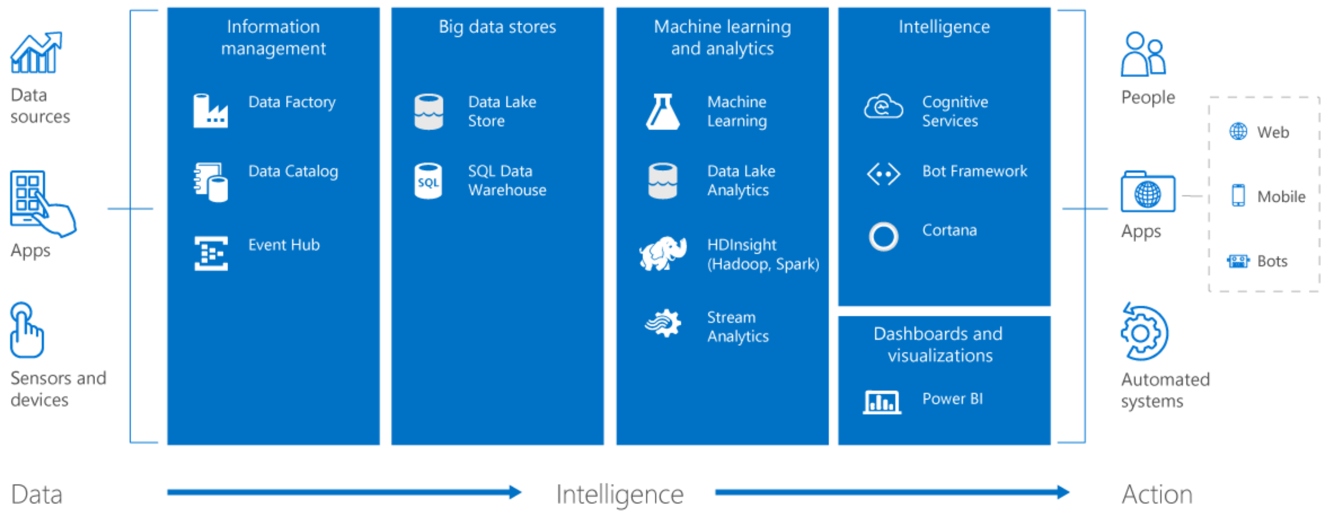


Figure 13. Cortana Intelligence Suite (Source: http://bit.ly/1r7dBXz)

While building our solution, we are mostly interested in Event Hub.

4.3.1. Azure Event Hub

Azure Event Hub is a highly scalable publish-subscribe service that can ingest millions of events per second and stream them into multiple applications. This allows to process and analyze the massive amounts of data produced by connected devices and applications. Event Hub is a managed service that ingests the events with an elastic scale to accommodate various load profiles, accordingly to use case. It meets the challenge of connecting disparate data sources while handling the scale of the aggregate stream. It provides the capacity to process events from millions of devices while preserving the order of the events on a per-device basis. Event Hub supports Advanced Message Queuing Protocol (AMQP) and HTTP, allowing many platforms to work with it. In our solution Event Hub will play a role of reliable pipeline between customer company and process mining tool (Figure 14).

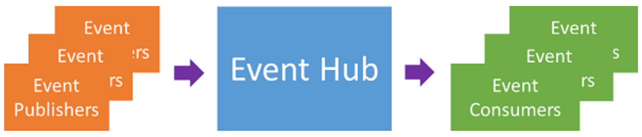


Figure 14. The role of Event Hub

Event Hub provides message streaming through partitioned consumer pattern, where the consumer only reads a specific subset or partition of the stream. This pattern enables horizontal scale for event processing. A partition is an ordered sequence of events that is held in an Event Hub. When a new event arrives, it is added to the end of the sequence (Figure 15):

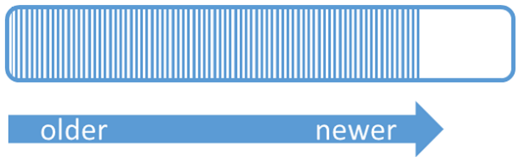


Figure 15. Partition in Event Hub

Partition retains data for a configured retention time that is set at the event Event Hub level. This setting applies to all partitions in the Event Hub. Events expire on a time basis and it not possible to explicitly delete them. Each partition is independent and contains its own sequence of data. The number of partitions can be specified at the Event Hub creation time and must be between 2 and 32. Partitions are a data organization mechanism and more related to the degree of downstream parallelism required in consuming applications that than to Event Hub throughput. This is directly related to the number of concurrent readers expected to have.

An entity that sends events to an Event Hub is an event publisher. Event publishers can publish events using either HTTPS or AMQP 1.0. They use Shared Access Signature (SAS) token to identify themselves to an Event Hub and can have a unique identity or use a common SAS token, depending on use case.

The common tasks for event publishers:

1) *Acquire SAS token*

   A SAS token is generated from a SAS key and is a SHA hash of a URL, encoded in a specific format. Using the name of the key (policy) and the token, Service Bus can regenerate the hash and thus authenticate the sender. Normally SAS tokens for event publishers are created with only send privileges on a specific Event Hub.

2) *Publish events*

   Service Bus provides an EventHubClient class for publishing events to an Event Hub from .NET clients. For other runtimes and platforms any AMQP 1.0 client can be used (e.g. Apache Qpid). Events can be published individually or batched. A single publication has a limit of 256 KB and publishing events larger than this limit results in error. The choice to use AMQP or HTTPS is specific to the usage scenario. AMQP requires the establishment of a persistent bidirectional socket in addition to transport level security (TLS) or SSL/TLS. This can be a costly operation in terms of network traffic but only happens at the beginning of an AMQP session. HTTP has a lower initial overhead but requires additional SSL overhead for every request. For publishers who frequently publish events, AMQP offers significant performance, latency and throughput savings.

In order to map incoming event data into specific partitions, a partition key is used. The partition key is a sender-supplied value passed into an Event Hub. It is processed through a static hashing function, the result of which creates the partition assignment (Figure 16). If partition key is not specified, a round robin assignment is used. Partition keys are important for organizing data for downstream processing but are fundamentally unrelated to partitions themselves. A per-device or user unique identity makes a good partition key, but other attributes such as geography can also be used to group related events into a single partition. Event Hubs ensures that any and all events sharing the same partition key value are delivered in order, and to the same partition.

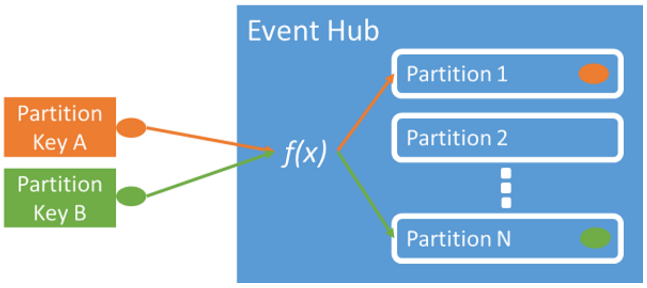


Figure 16. The stream partitioning

Any entity that reads event data from an Event Hub is an event consumer. All event consumers read the event stream through partitions in a consumer group. Each partition should have only one active reader at a time. All Event Hubs consumers connect via the AMQP 1.0 session, in which events are delivered as they become available. The client does not need to poll for data availability.

The publish/subscribe mechanism of Event Hubs is enabled through consumer groups. A consumer group is a view (state, position, or offset) of an entire Event Hub. Consumer groups enable multiple consuming applications to each have a separate view of the event stream, and to read the stream independently at their own pace and with their own offsets. In a stream processing architecture, each downstream application equates to a consumer group. If you want to write event data to long-term storage, then that storage writer application is a consumer group. Complex event processing is performed by another, separate consumer group. Partitions can be accessed only through a consumer group. There is always a default consumer group in an Event Hub, and up to 20 consumer groups can be created for a Standard tier Event Hub.

The common tasks for event consumers:

1) Connect to a partition

In order to consume events from an Event Hub, a consumer must connect to a partition through a consumer group. As part of the partitioned consumer model, only a single reader should be active on a partition at any one time within a consumer group. It is common practice when connecting directly to partitions to use a leasing mechanism in order to coordinate reader connections to specific partitions. This way, it is possible for every partition in a consumer group to have only one active reader. Managing the position in the sequence for a reader is an important task that is achieved through checkpointing. This functionality is simplified by using the EventProcessorHost class for .NET clients. EventProcessorHost is an intelligent consumer agent and is described in the next section.

2) *Read events*

After an AMQP 1.0 session and a link ares opened for a specific partition, events are delivered to the AMQP 1.0 client by the Event Hubs service. This delivery mechanism enables higher throughput and lower latency than pull-based mechanisms such as HTTP GET. As events are sent to the client, each event data instance contains important metadata such as the offset and a sequence number that is used to facilitate checkpointing on the event sequence.

The throughput capacity of Event Hubs is controlled by throughput units. Throughput units are pre-purchased units of capacity. A single throughput unit includes the following:

1) Ingress: Up to 1 MB per second or 1000 events per second
2) Egress: Up to 2 MB per second

Ingress is throttled to the amount of capacity provided by the number of throughput units purchased. Sending data above this amount results in a "quota exceeded" exception. This amount is either 1 MB per second or 1000 events per second, whichever comes first. Egress does not produce throttling exceptions but is limited to the amount of data transfer provided for by the purchased throughput units: 2 MB per second per throughput unit.

4.3.2. Visualization

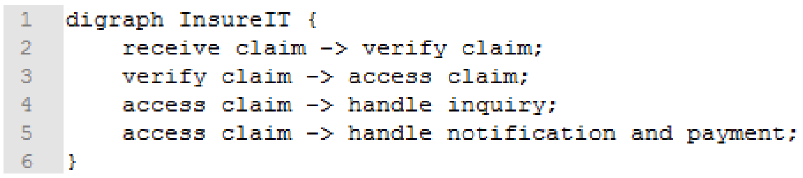
In addition to Azure Event Hub and Apache Storm, an open source graph visualization software called Graphviz will be used. It is a powerful tool, working with special algorithms to render the best graph possible. Graphviz is a command-line tool, which means that graph can be generated just from one simple command, a huge text file or automatically from data generated by another process. The Graphviz layout programs take descriptions of graphs in a simple text language and make diagrams in useful formats, such as images and SVG for web pages; PDF or Postscript for inclusion in other documents; or display in an interactive graph browser. Graphviz has many useful features for concrete diagrams, such as options and colors, fonts, tabular node layouts, line styles, hyperlinks and custom shapes. In general, Graphviz is a package, which consists of a graph description language named DOT and a set of tools to process DOT files:

1) *dot* – command-line tool to draw hierarchical directed graphs
2) *neato* – makes layouts of undirected graphs following the filter model of DOT. It is used in case the graph is not large (about 100 nodes).
3) *fdp* – similar to *neato,* but draws the graphs by reducing forces rather than working with energy
4) *sfdp* – multiscale version of *fdp* for drawing of the large graphs
5) *twopi* – tool for radial graph layouts
6) *circo* – tool for circular graph layouts
7) *dotty* – a graphical user interface to visualize and edit graphs
8) *lefty* – a programmable widget that displays DOT graphs and allows the user to perform actions on them with the mouse. It can be used as the view in the model-view-controller GUI applications that use graphs

Taking into consideration the definition of process map the *dot* tool should be used.

Dot draws directed graphs as hierarchies. It runs as a command line program, web visualization service, or with a compatible graphical interface. Its features include well-tuned layout algorithms for placing nodes and edge splines, edge labels, “record” shapes with “ports” for drawing data structures, cluster layouts and an underlying file language for stream-oriented graph tools. Dot draws a graph in four main phases. The layout procedure used by dot relies on the graph being acyclic. Therefore, the first step is to break any cycles which occur in the input graph be reversing the internal direction of certain cyclic edges. The next step assigns nodes to discrete ranks or levels. In a top-to-bottom drawing, ranks determine Y coordinates. Edges that span more than one rank are broken into chains of “virtual” nodes and unit-length edges. The third step orders nodes within ranks to avoid crossings. The fourth step sets X coordinates of nodes to keep edges short, and the final step routes edge splines.

Dot accepts input in the DOT language. This language describes three kinds of objects: graphs, nodes, and edges. A node is created when its name first appears in the file. An edge is created when nodes are joined by the edge operator ->. Figure 17 shows an example of dot-file:



```
digraph InsureIT {
    receive claim -> verify claim;
    verify claim -> access claim;
    access claim -> handle inquiry;
    access claim -> handle notification and payment;
}
```

Figure 17. Small graph

Running command dot on this file will result in the following graph (Figure 18):

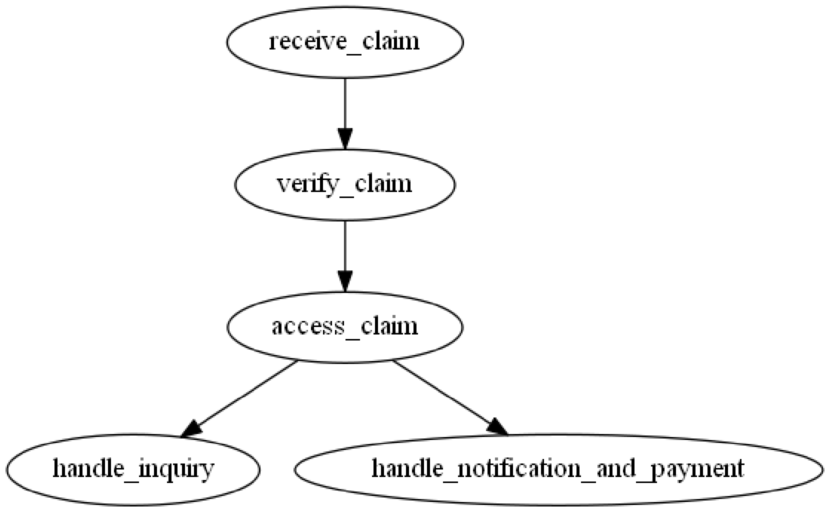


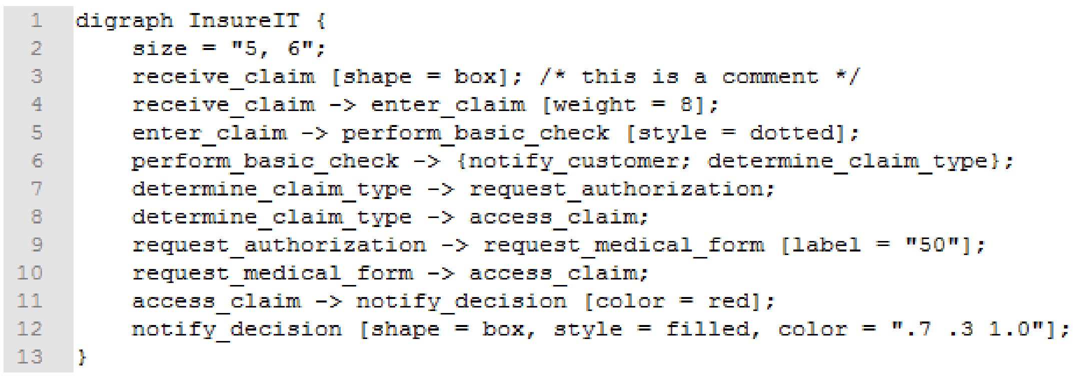
Figure 18. Drawing of the small graph

The full command to create this graph looks as follows:

$ dot -Tps graph1.dot -o graph1.ps

-Tps specifies the output format (Postscript in this case), graph1.dot – input file and graph1.ps – output file.

It is possible to adjust the representation or placement of nodes and edges in the layout, by setting attributes of the elements in the input file. Attributes are name-value pairs of character strings. In the listing of Figure 19, line 2 sets the graph’s size to 5, 6 (in inches). If the drawing is too large, it is scaled as necessary to fit. Nodes or edges attributes are set off in square brackets. In line 3, the node *receive_claim* is assigned shape box. The edge in line 4 is straightened by increasing its weight. The edge in line 5 is drawn as a dotted line. Line 6 makes edges from *perform_basic_check* to *notify_customer* and *determine_claim_type*. Line 9 makes an edge labeled 50. In line 11 the default edge color is set to red. Label 12 changes the default node to be a box filled with a shade of blue.



```
digraph InsureIT {
    size = "5, 6";
    receive_claim [shape = box]; /* this is a comment */
    receive_claim -> enter_claim [weight = 8];
    enter_claim -> perform_basic_check [style = dotted];
    perform_basic_check -> {notify_customer; determine_claim_type};
    determine_claim_type -> request_authorization;
    determine_claim_type -> access_claim;
    request_authorization -> request_medical_form [label = "50"];
    request_medical_form -> access_claim;
    access_claim -> notify_decision [color = red];
    notify_decision [shape = box, style = filled, color = ".7 .3 1.0"];
}
```

Figure 19. Fancy graph

The processing of this file will result in the following graph (Figure 20):

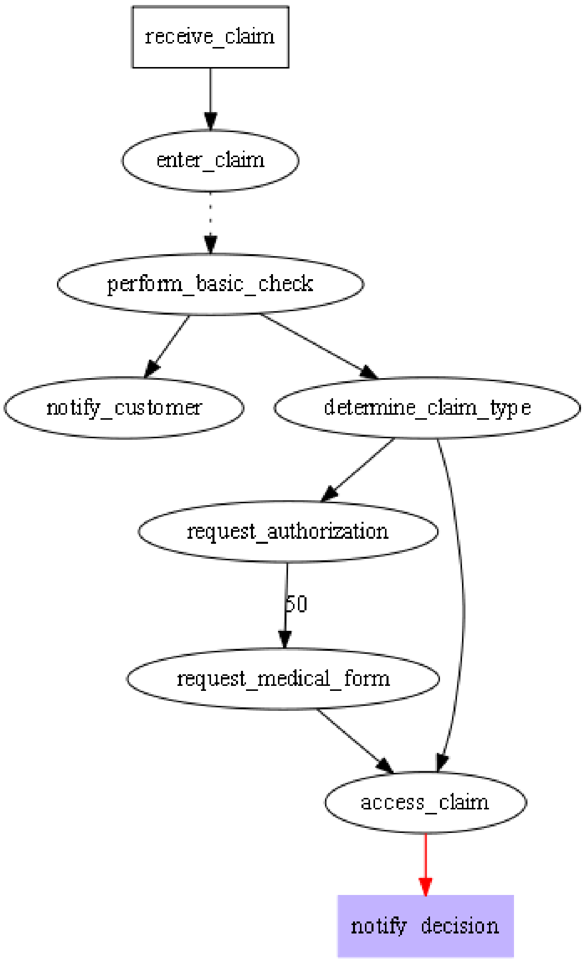


Figure 20. Drawing of the fancy graph

4.3.3. Summary

The architecture of the solution considering the implementation details will look as follows (Figure 21):

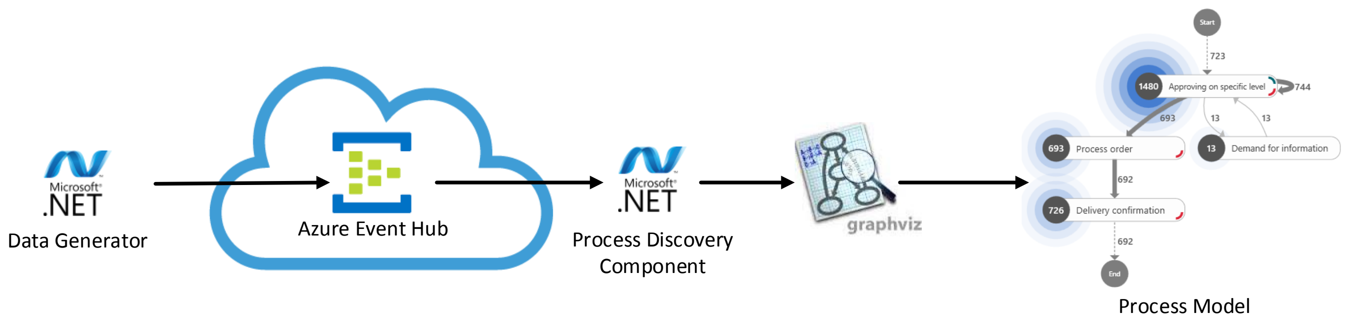


Figure 21. Implementation details of the solution

Data Generator is the component which simulates the customer company. It will continuously send the data to our system. Event Hub will play a role of a reliable channel between the customer and our application. It acts as spout (source) for Process Discovery Component, which performs the frequency counting operation based on received data. The main role of Graphviz is to discover the control-flow of the process and to represent it in the form of a process map.

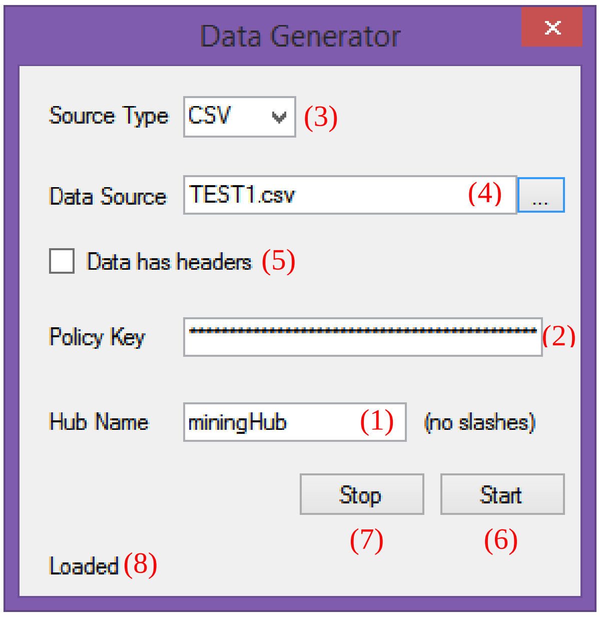
5 Validation

In this section, we will discuss the functionality of the system and conduct the set of experiments to validate the theoretical results.

5.1 System description

Before we start the experiments, it is important to understand how the system works and what data do we need to run it properly.

The system consists of two modules: data generator (Figure 22), which simulates sending of the data from an external company and process discovery module (Figure 23), which is responsible for process discovery and analysis. They communicate through Azure event hub and data storage account. Therefore, event hub name and storage account name along with corresponding policy and account keys are crucial data to establish the communication. When sending the data we need event hub name (1) and its policy key (2). Since there are many types of event log files (e.g. CSV, MXML, XES), there should be different parsers to get all necessary components for process discovery, such as case id, activity name, timestamp and activity type. Therefore, before processing the data, we need to specify the source type (3) in order to trigger correct parser. Finally, we have to choose the data source (4) and specify whether it has headers (5). To start to send the data we need to press Start button (6) and we can pause this procedure by pressing Stop button (7). All actions (e.g. reading of data source, sending of the data, etc.) will be reflected on status notifier (8).


Data Generator

Source Type: CSV (3)

Data Source: TEST1.csv (4)

☐ Data has headers (5)

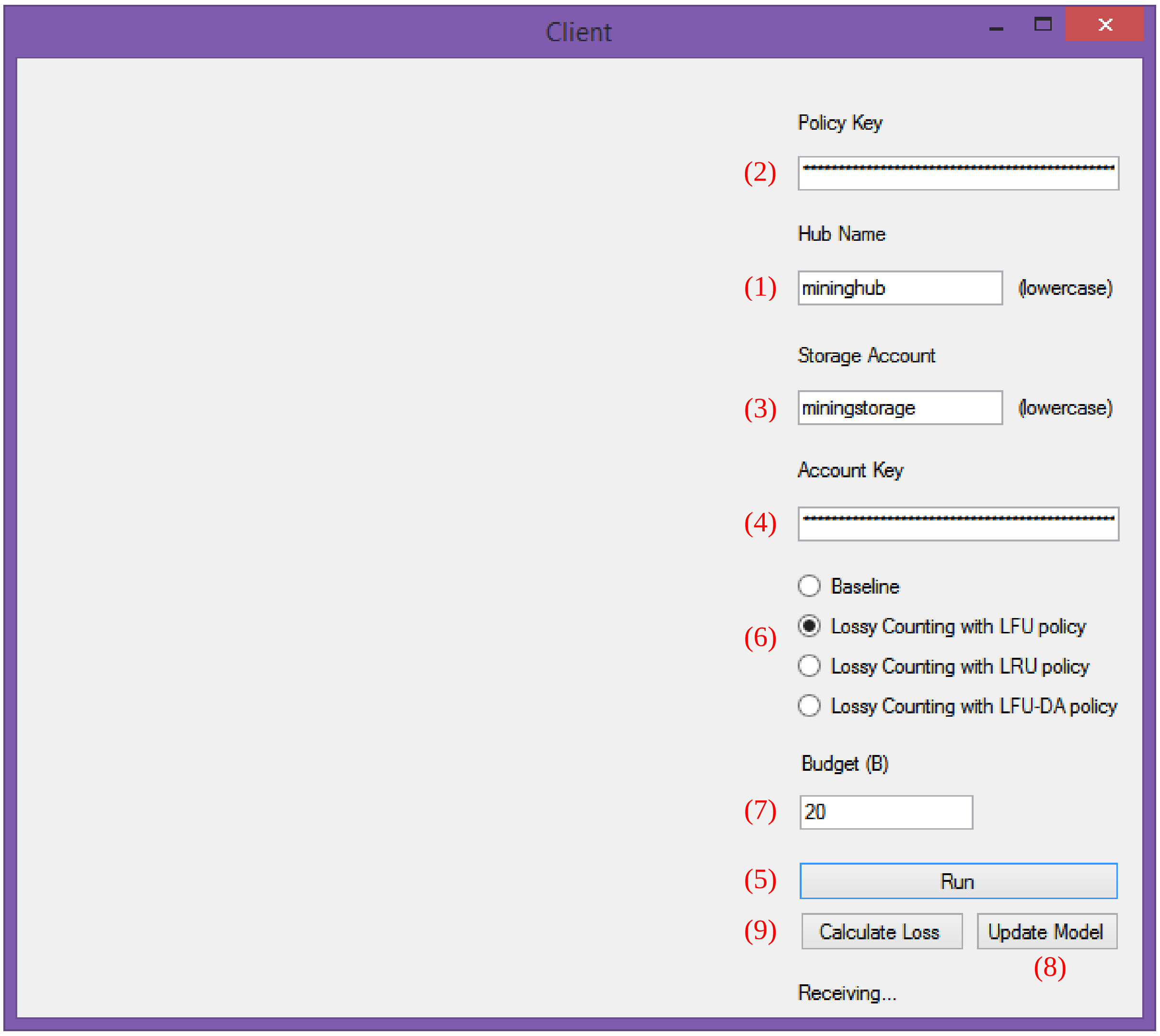
Policy Key: ** (2)

Hub Name: miningHub (1) (no slashes)

Loaded (8)

Figure 22. Data generator

For receiving the data from event hub, we need hub name (1) and policy key (2), storage account name (3) and storage account key (4). To any event hub, storage account can be assigned, and it is used to access the data in event hub. We can start to listen to storage account by pressing Run button (5). However, before a start of the receiving information, we have to choose process discovery method (6) and specify the budget (7). With occurrence of a new element, chosen process discovery technique will be applied in order to update the current state of a process. After finishing listening, we can visualize discovered model by pressing Update Model button (8). The resulting model will be shown on the left part of the form. In addition, it is possible to estimate the model loss by using Calculate Loss button (9).


Client

Policy Key

(2) **

Hub Name

(1) mininghub (lowercase)

Storage Account

(3) miningstorage (lowercase)

Account Key

(4) **

(6)
- ( ) Baseline
- (●) Lossy Counting with LFU policy
- ( ) Lossy Counting with LRU policy
- ( ) Lossy Counting with LFU-DA policy

Budget (B)

(7) 20

(5) Run

(9) Calculate Loss | Update Model (8)

Receiving...

Figure 23. Process Discovery Component

5.2 Experimental Evaluation

In the evaluation, we want to answer the following question:

Can we achieve better accuracy-memory trade-off using three proposed approaches relative to currently existing methods?

The following event logs were used in the evaluation:

1) BPI Challenge 2017.csv
   Number of traces: 31509
   Number of events: 561671
   Events per trace: 17.825
   Number of distinct event labels (activities): 26

   This event log describes a loan application process of a Dutch financial institute. It contains all applications filed through an online system in 2016 and their subsequent events until February 1st 2017, 15:11. It has the following attributes:

Case.ID, Activity, Resource, Start.Timestamp, Complete.Timestamp, X.case.ApplicationType, X.case.LoanGoal, X.case.RequestedAmount, Accepted, Action, CreditScore, EventID, EventOrigin, FirstWithdrawalAmount, MonthlyCost, NumberOfTerms, OfferID, OfferedAmount, Selected and lifecycle.transition.

2) BPI Challenge 2016 – Clicks Logged In.csv (doi:10.4121/uuid:01345ac4-7d1d-426e-92b8-24933a079412)
   Number of traces: 660270
   Number of events: 7174934
   Events per trace: 10.866
   Number of distinct event labels (activities): 600

   The dataset consists of the following attributes:
   CustomerID, AgeCategory, Gender, Office_U, Office_V, SessionID, IPID, Timestamp, Vhost, URL_File, Page_Name, REF_URL_Category, page_load_error, page_action_detail, tip, service_detail, xps_info page_action_detail_EN, service_detail_EN, tip_EN.

3) Cloud Invoice Approval Anonymized.csv (a log provided by Minit company)
   Number of traces: 5330
   Number of events: 66074
   Events per trace: 12.39662
   Number of distinct event labels (activities): 21

   This dataset describes the invoice approval process of an unknown company. It consist of the following attributes: CaseID, Activity, StartTimestamp, EndTimestamp, EventType, ActivityStatus, EventStatus, Resource, ActivityFinalAction, CaseStatus, CostCenter.Code, InvoiceStatus.DisplayName, InvocieTotalAmountWithoutVAT, Supplier.City, Supplier.Name and Supplier.State.

In order to use these event logs, pre-processing has to be done first. The events of every log have to be sorted in chronological order. Only the necessary attributes will be left and event type attribute will be added in order to keep track of running cases. All of the pre-processing changes are described below:

1) BPI Challenge 2017.csv
   After pre-processing, this log will have the following attributes: Case.ID, Activity, Start.Timestamp and Event.Type. In addition, for testing purposes, the size of the log was reduced to 52033 events. The characteristics of the log:
   Number of traces: 2862
   Number of events: 52033
   Events per trace: 18.1806
   Number of distinct event labels (activities): 25

2) BPI Challenge 2016 – Clicks Logged In.csv
   Here, SessionID was treated as case identifier, and it was assumed that page name represents activity. In addition, the Event.Type attribute was added to keep the running cases. Since the log is very large, its size was reduced to 100,000 events for simplicity of testing. The characteristics of the log:
   Number of traces: 7857
   Number of events: 100000
   Events per trace: 12.727
   Number of distinct event labels (activities): 198

3) Cloud Invoice Approval Anonymized.csv
   After all the elements had been ordered by timestamp, the Event.Type attribute was added. In addition, only relevant attributes were left. Therefore, the final log has the following attributes: Case.ID, Activity, Timestamp and Event.Type. Finally, the size of the log was reduced to 6726 events. The characteristics of the dataset:
   Number of traces: 529
   Number of events: 6726
   Events per trace: 12.714
   Number of distinct event labels (activities): 21

The events of each dataset were streamed by the Data Generator and sent to the Process Discovery component. Such metrics as a model loss and memory load were measured during experiments. Since the speed of processing is highly dependent on the internet conditions, it was ignored and we did not calculate this metric. During all the experiments, the budget was used as an independent variable. It was used to limit the discovery process in terms of memory usage and model size.

Since the most efficient existing online process discovery method is Lossy Counting with Budget (LCB), it was taken as a baseline and three its proposed modifications such as LCB with LRU policy, LCB with LFU policy and LCB with LFU-DA policy were evaluated.

In order to answer the question specified in the beginning of this subsection, we need to introduce two measures: accuracy and memory consumption.

Accuracy shows the level of similarity between original and discovered models. Accuracy is derived from loss, which is expressed by the total difference of frequencies between original and produced models. The idea is to treat missing arcs as arcs with a frequency of 0. If an arc (relation) with frequency n exists in the actual process map but it does not exist in the map produced by the system, it has to be “conceptually” added to the map obtained by the system with a frequency 0. The difference between the frequencies will be equal to $|n-0| = n$. If vice-versa, an arc exists in the discovered map, but not in the actual model, then similarly, we make this arc exist with frequency 0. If an activity is missing in the discovered model but it exists in the original map, then it is added to produced model and all relations coming out or leading to this activity receive the frequency of 0 (and vice versa). Figure 24 shows actual and discovered models and underlines their difference. We can see that “Determine Claim Type” → “Request Medical Form” arc is missing in the discovered model. Therefore, we add its frequency to the loss. In addition, obtained model does not include activity “Notify Customer”. Since it has just one incoming relation (“Perform Basic Check” → “Notify Customer”), the frequency of this arc has to be added to the loss. The total loss is equal 2.

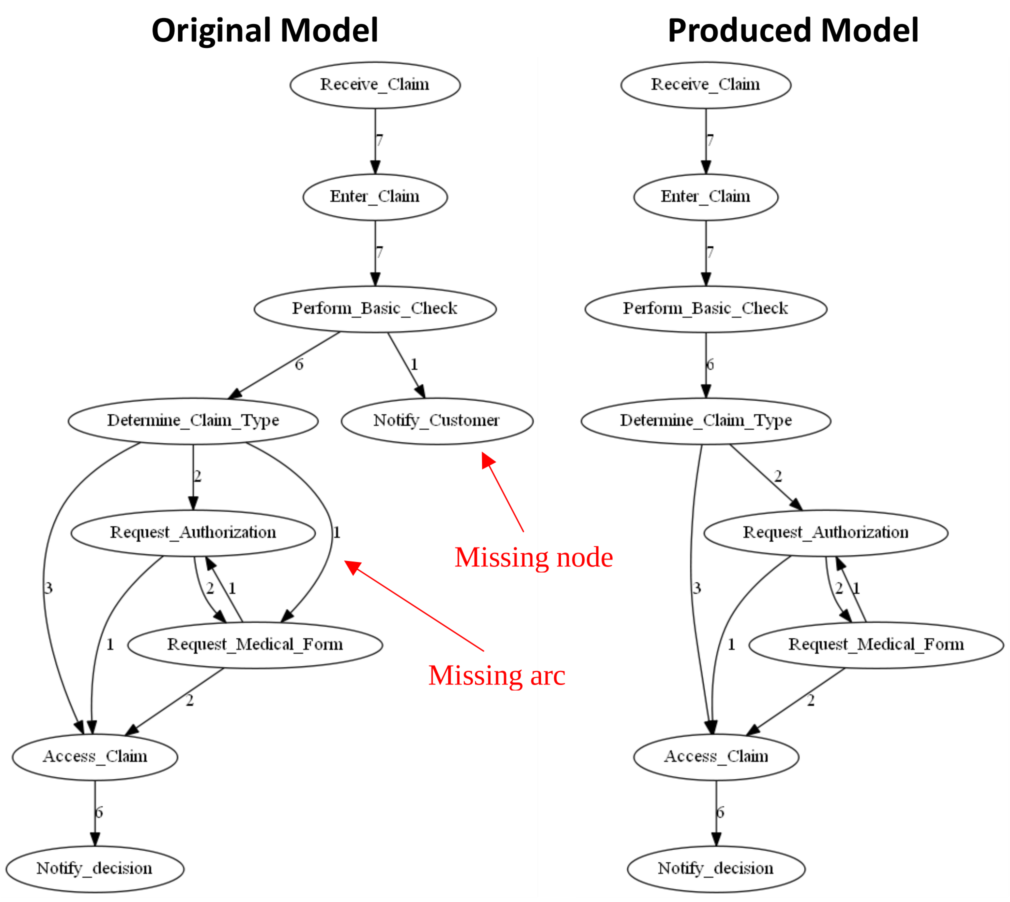


Figure 24. Calculation of the loss

The accuracy (A) now can be calculated as:

$$A = 1 - \frac{Loss}{Total\ Frequency} \quad (9)$$

In the example above, A will be equal to:

$$A = 1 - \frac{2}{39} \approx 0.94$$

Memory consumption shows the amount of memory used during the discovery process. It is measured in the terms of memory words, which refer to a fixed-size unit of memory, enough to store one identifier or a number. All of the algorithms has different memory consumption. For LCB algorithm memory consumption for one case, activity or relation objects is the following:

$$For\ LCB: \begin{cases} M(activity) = M(name, frequency, bcur) = 3 \\ M(relation) = M(startActivity, endActivity, frequency, bcur) = 4 \\ M(case) = M(caseID, lastActivity, frequency, bcur) = 4 \end{cases}$$

$$For\ LFU: \begin{cases} M(activity) = M(name, frequency) = 2 \\ M(relation) = M(startActivity, endActivity, frequency) = 3 \\ M(case) = M(caseID, lastActivity, type) = 3 \end{cases}$$

$$For\ LRU: \begin{cases} M(activity) = M(name, frequency, lastSeenIndex) = 3 \\ M(relation) = M(startActivity, endActivity, frequency, lastSeenIndex) = 4 \\ M(case) = M(caseID, lastActivity, type) = 3 \end{cases}$$

$$For\ LFU-DA:\begin{cases}M(activity)=M(name,frequency,L)=3\\M(relation)=M(startActivity,endActivity,frequency,L)=4\\M(case)=M(caseID,lastActivity,type)=3\end{cases}$$

The total memory consumption for a certain method can be calculated by the next formula:

$$M=M(activity)*N_a+M(relation)*N_r+M(case)*N_c \quad (10)$$

where N_a, N_r, N_c – number of activities, relations and cases currently in memory

The evaluation of the methods was done using two logs: Cloud Invoice Approval Anonymized.csv (event log 1) and BPI Challenge 2016 – Clicks Logged In.csv (event log 2). The results of the evaluation presented below.

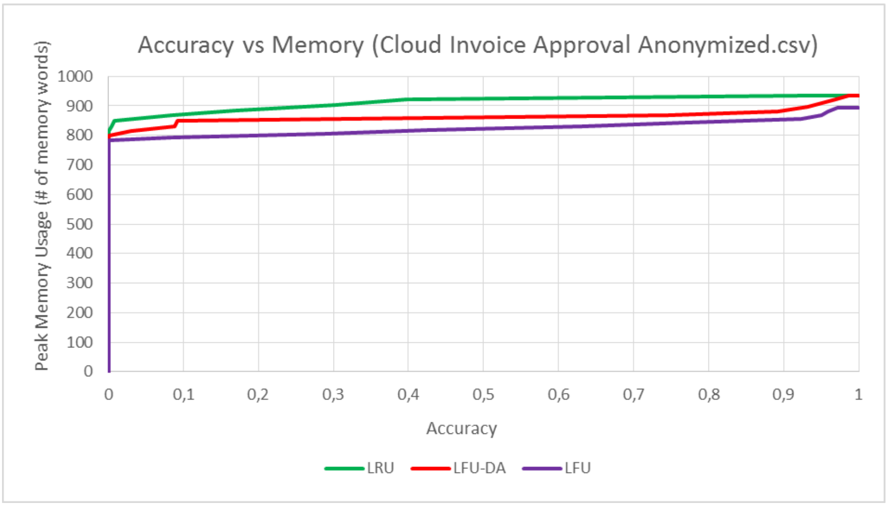


Figure 25. Comparison of proposed approaches (event log 1)

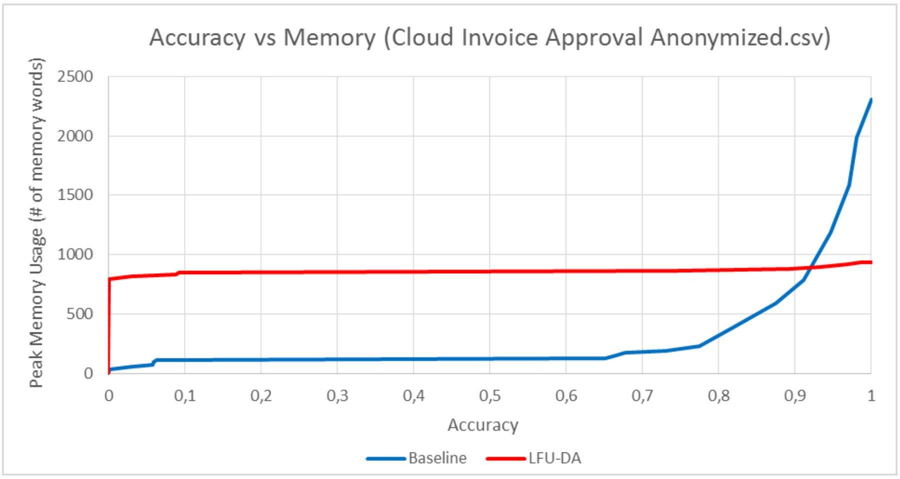


Figure 26. Comparison of LFU-DA with the baseline (event log 1)

On Figure 25, three proposed approaches were compared with respect to their accuracy and memory consumption. LFU approach showed the best results. It is the best option in case the process is stable. However, it can not deal with concept drifts. LFU-DA algorithm can solve this problem. It has slightly larger memory consumption because it also keeps the recency values for objects. LFU-DA can be considered as an alternative to LCB, therefore both approaches were compared in the Figure 26. The memory consumption for LFU-DA is more or less constant on any level of accuracy, while LCB needs more memory to provide more accurate results. Although LCB consumes less memory to get the model with accuracy up to 0.9, LFU-DA produces the lossless model with a smaller amount of memory needed.

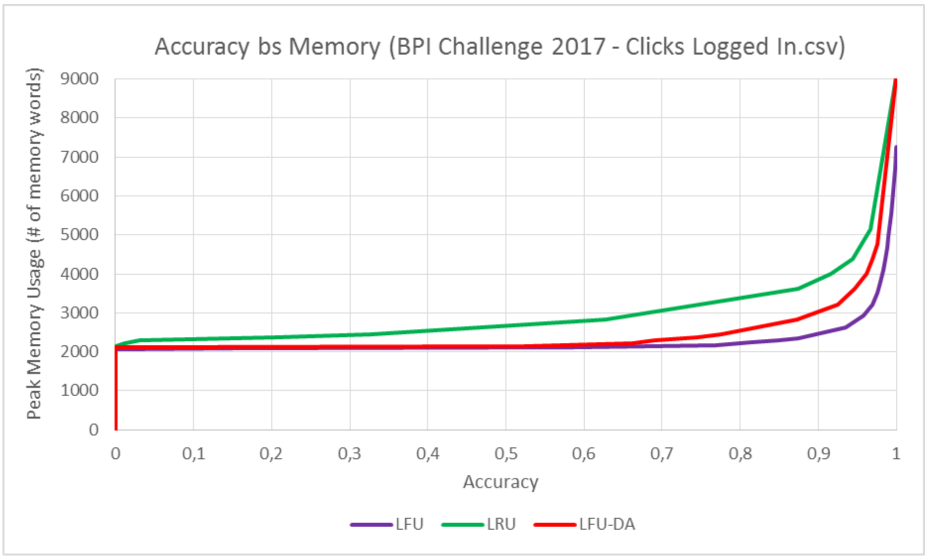


Figure 27. Comparison of proposed approaches (event log 2)

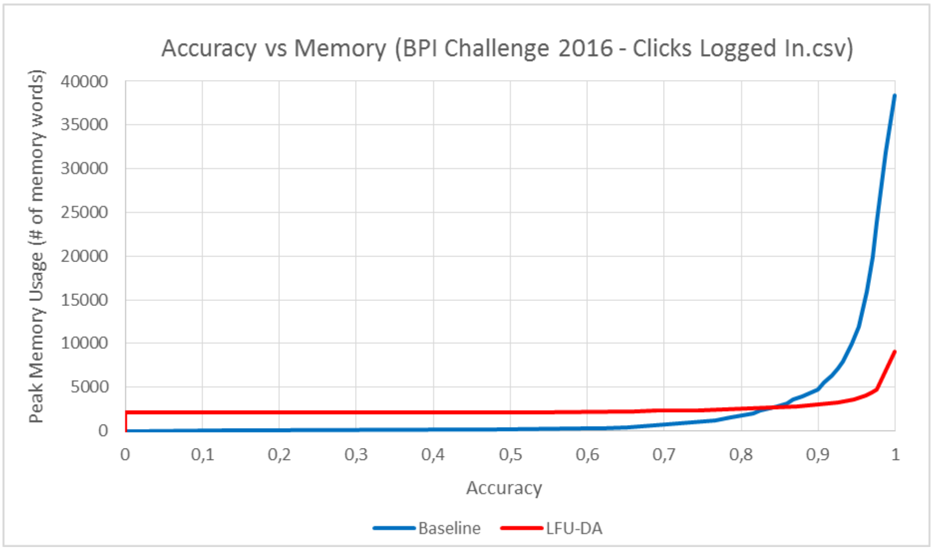


Figure 28. Comparison of LFU-DA with baseline (event log 2)

The general behavior of LFU-DA and LCB is the same as it was shown in Figure 26. However, the difference in memory consumption is significantly larger (Figure 28). This is because the first log (Cloud Invoice Approval Anonymized.csv) consists of 529 cases and 21 distinct activities, while the second one (BPI Challenge – Clicks Logged In.csv) – 7857 cases and 198 distinct activities. Since LCB is case dependent, it needs a much larger budget to deal with the second log. In addition, the number of running cases in a certain time is not high for clicks data. This strongly affects LFU-DA, because it keeps running cases. In the same time, LCB does not care whether a case is running or already finished, keeping all of them and deleting when needed. Therefore, we can say that when dealing with huge processes with high variance (a large number of distinct traces and activities), LFU-DA is much more efficient algorithm than LCB. With an increase of the size of the process (and a number of cases), the difference in memory consumption is getting larger and larger, thus we can achieve better accuracy-memory tradeoffs applying proposed approaches.

The evaluation was done using only two event logs, and hence the conclusions might not generalize to other event logs. In addition, we focused on accuracy with respect to only one measure.

5.3 Qualitative Results

Most of the time we want to get lossless model because the output of the system will be then sent to the systems with filtering mechanisms. However, the proposed approach also has a potential to filter the event log. BPI Challenge 2017 log is taken as an example. Model discovered from this log without filtering is spaghetti-like and unreadable (Figure 29).

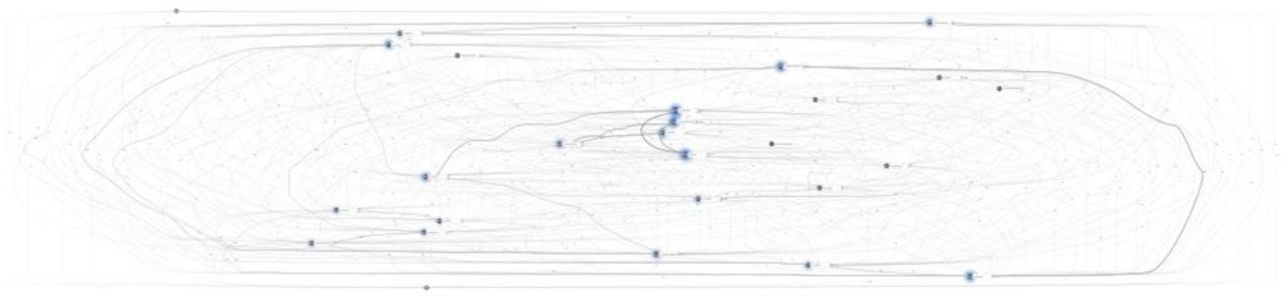
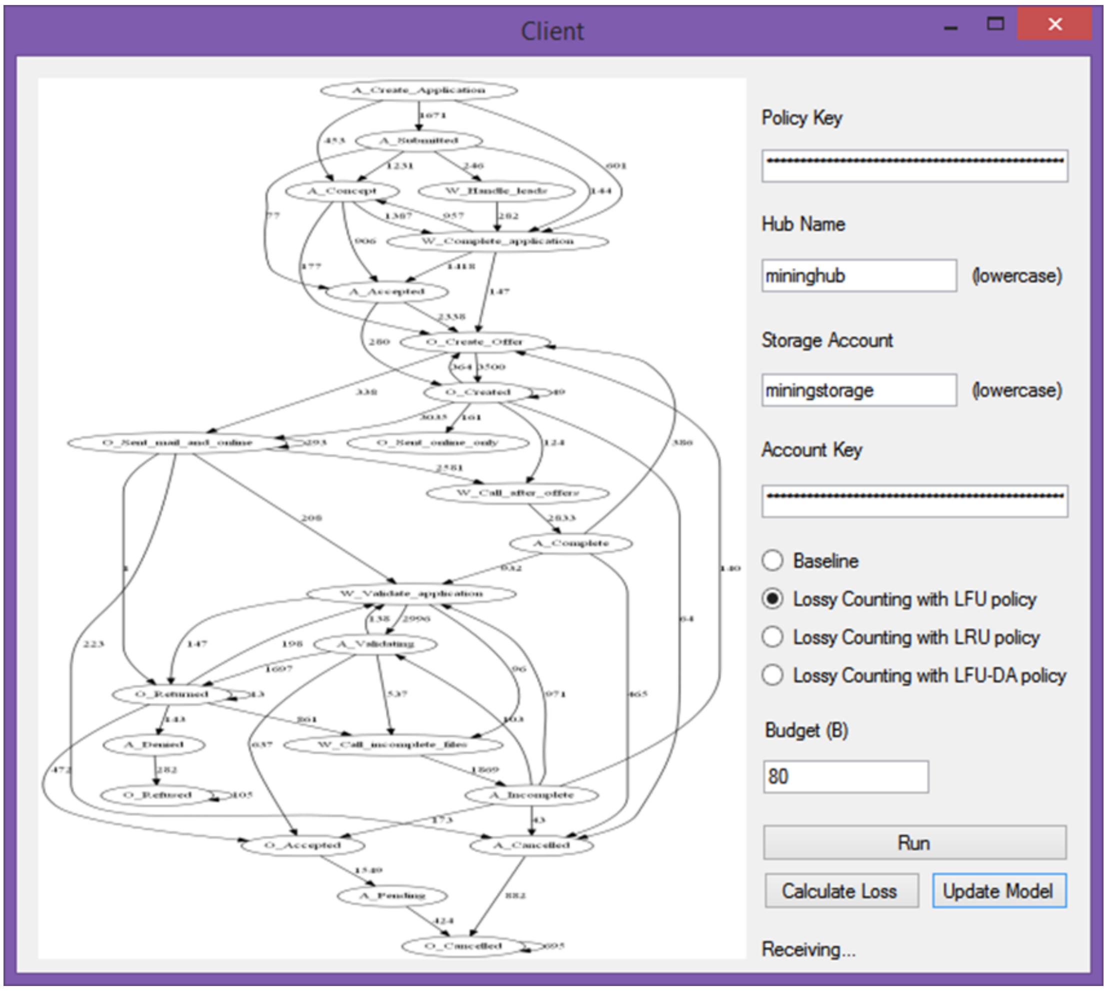


Figure 29. Model of loan application process (from BPIC 2017)

Let us try to discover this model using our system. For this, Lossy Counting with LFU policy algorithm was used and the budget was set to 80. Figure 30 presents results of process discovery:



Policy Key

••

Hub Name

mininghub (lowercase)

Storage Account

miningstorage (lowercase)

Account Key

••

○ Baseline
● Lossy Counting with LFU policy
○ Lossy Counting with LRU policy
○ Lossy Counting with LFU-DA policy

Budget (B)

80

Receiving...

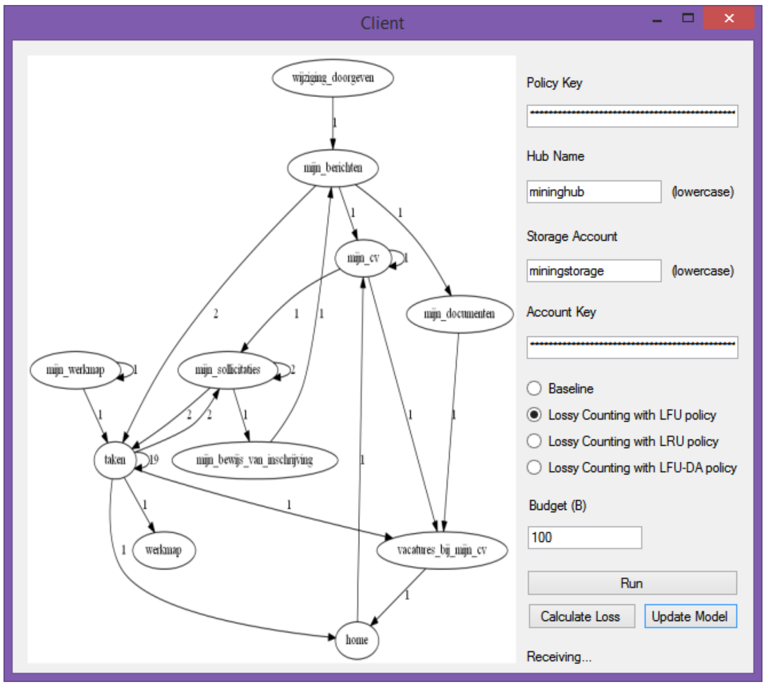
Figure 30. Model of loan application process obtained by system

As result, we have obtained filtered model, which represent only highly frequent behavior.

1) Dealing with huge processes

Sometimes event logs can be huge enough to cause computational and memory issues. Since most of the process discovery techniques are log-oriented and process data in batches, they are vulnerable to such kind of problem. Online discovery deals with this problem very well, because it processes incoming data element-wise. This can be seen by taking Clicks data from BPIC 2016 as an example. Let us set budget to 100 and show process model after 50, 100, 1000 and 10000 events

Process model after 50 events:



Policy Key
••

Hub Name
mininghub (lowercase)

Storage Account
miningstorage (lowercase)

Account Key
••

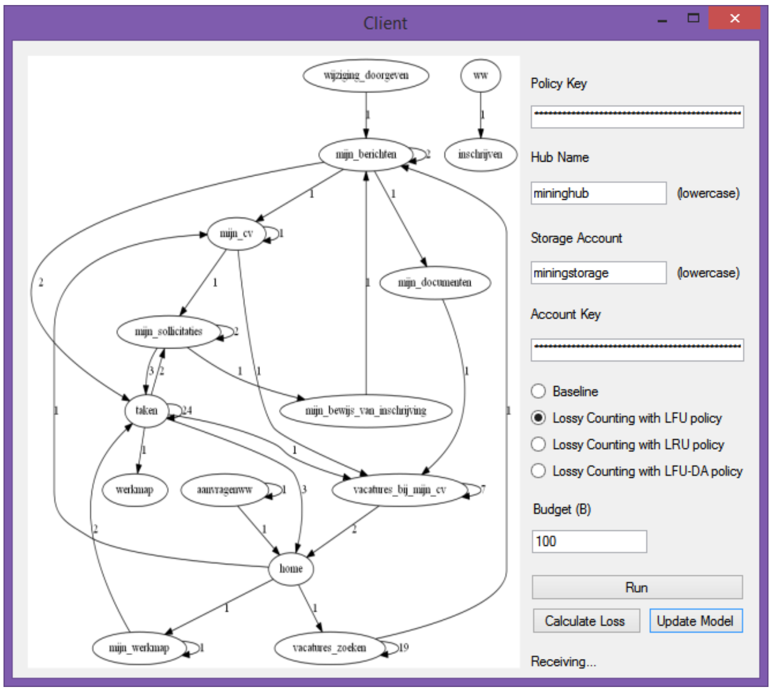
Baseline
[✓ Lossy Counting with LFU policy]
Lossy Counting with LRU policy
Lossy Counting with LFU-DA policy

Budget (B)
100

Run
Calculate Loss
Update Model
Receiving...

Figure 31. Process model after 50 events (B = 100)

Process model after 100 events:



Policy Key
••

Hub Name
mininghub (lowercase)

Storage Account
miningstorage (lowercase)

Account Key
••

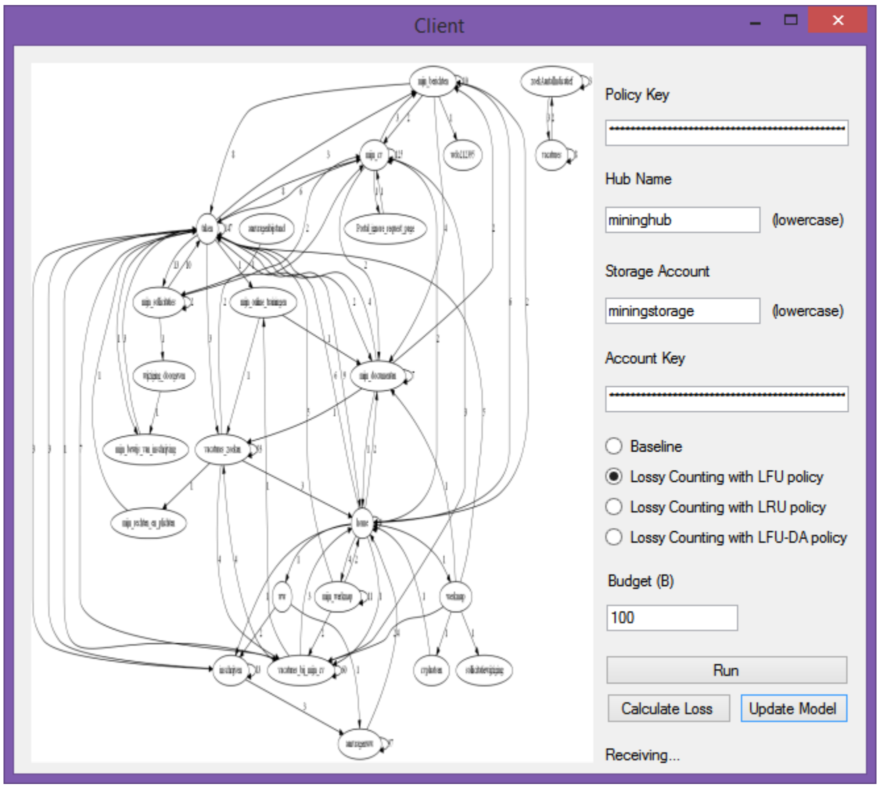
Baseline
[✓ Lossy Counting with LFU policy]
Lossy Counting with LRU policy
Lossy Counting with LFU-DA policy

Budget (B)
100

Run
Calculate Loss
Update Model
Receiving...

Figure 32. Process model after 100 events (B = 100)

Process model after 1000 events:


Client

Policy Key: ••

Hub Name: mininghub (lowercase)

Storage Account: miningstorage (lowercase)

Account Key: ••

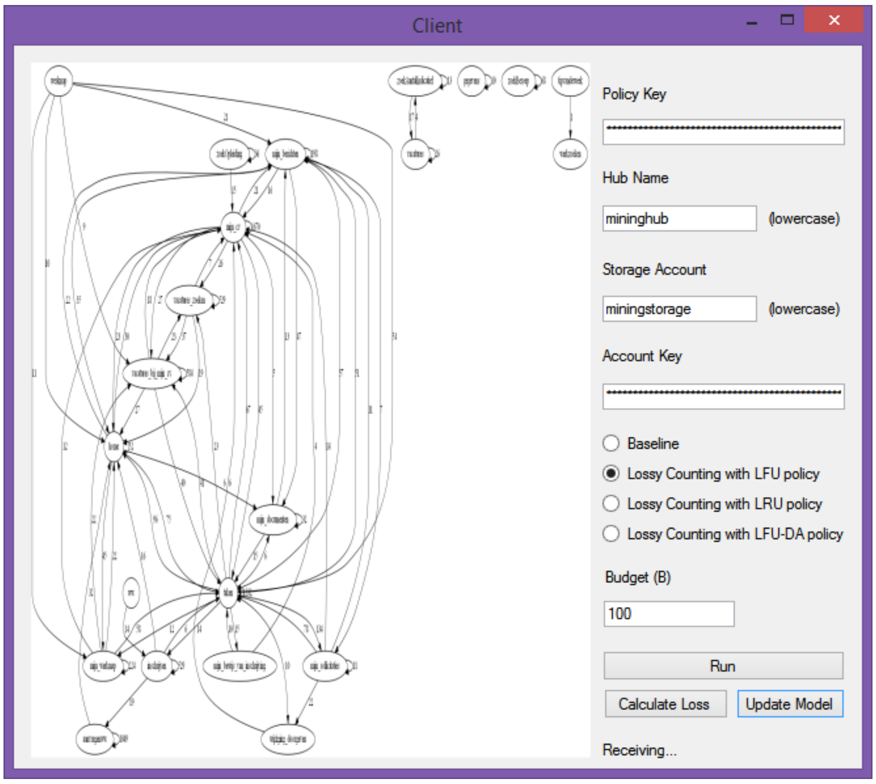
- ○ Baseline
- ● Lossy Counting with LFU policy
- ○ Lossy Counting with LRU policy
- ○ Lossy Counting with LFU-DA policy

Budget (B): 100

Receiving...

Figure 33. Process model after 1000 events (B = 100)

Process model after 10000 events:


Client

Policy Key: ••

Hub Name: mininghub (lowercase)

Storage Account: miningstorage (lowercase)

Account Key: ••

- ○ Baseline
- ● Lossy Counting with LFU policy
- ○ Lossy Counting with LRU policy
- ○ Lossy Counting with LFU-DA policy

Budget (B): 100

Receiving...

Figure 34. Process model after 10000 events (B = 100)

6 Conclusion and Future Work

In the thesis, we have applied cache management methods to incremental process discovery. As a result, we have created an efficient online discovery algorithm based on LCB algorithm with an addition of cache replacement policies. Depending on the use case, we can use either LFU or LFU-DA policies to discover and analyze the process. LFU policy would be good when working with a constant and stable process. When dealing with a process that is changing in time, LFU-DA policy is a better option. We eliminated the concept of buckets in Lossy Counting with Budget algorithm and gave a more practical interpretation to budget. In addition, we have introduced the notion of activity type to keep track of running cases and remove finished ones. We have shown that incremental process discovery is a powerful mechanism to work with huge and complex processes. Instead of batch processing where the memory efficiency depends on the size of an event log, online discovery process data element-wise, providing actual and "fresh" model of the process.

Based on the results of our experiments we can say that the proposed approach is able to achieve low loss with considerably less memory usage than the LCB baseline. If the number of distinct events in the log is known beforehand, it is possible to set the budget in a way that ensures the lossless discovery of the process map, whereas this is not possible with the baseline LCB approach.

Due to the limitations of the prototype, it was not possible to conduct an experiment with the full BPI Challenge 2016 log, but instead, the first 100K events were taken. Once the scalability limitations of the current prototype are overcome, it will be possible to conduct experiments with larger event logs. Currently, the system is implemented in a way that all computations are done on the client side. The server plays a role as a secure and reliable channel between an external company and client system. It takes the events and sends them to the client. The current architecture is not sufficiently scalable because it requires all events to be streamed from the server to the client. Therefore, it would be good to move all computations to the cloud and for this, Apache Storm can be used.

7 References

[1] Burattin, A., Cimitile, M., Maggi, F. M. & Sperduti, A. (2015). Online discovery of declarative process models from event streams. *IEEE Trans. Services Computing*, vol. 8, no. 6, pp. 833-846.

[2] Van der Aalst, W. M. P. (2011). Process Mining – Discovery, Conformance and Enhancement of Business Processes. Netherlands/Eindhoven: Springer Berlin Heidelberg.

[3] Ekanayake C.C., Dumas M., García-Bañuelos L. & La Rosa M. (2013) Slice, Mine and Dice: Complexity-Aware Automated Discovery of Business Process Models. In: Daniel F., Wang J., Weber B. (eds) Business Process Management. Lecture Notes in Computer Science, vol 8094. Springer, Berlin, Heidelberg

[4] Anderson, C. (2013) What is a Process Map? Make a Process Map, accessed August 3, 2013 from: https://www.bizmanualz.com/make-a-process-map/what-is-a-process-map.html

[5] Burattin, A., Sperduti, A. & van der Aalst, W. M. P. (2014). Control-flow discovery from event streams. IEEE Congress on Evolutionary Computation 2014, pp. 2420-2427.

[6] Gupta, E. P. (2014). Process Mining A Comparative Study. International Journal of Advanced Research in Computer and Communication Engineering, vol. 3(11).

[7] Burattin, A., Sperduti, A., & van der Aalst, W. M. P. (2012). Heuristics Miners for Streaming Event Data. arXiv:1212.6383.

[8] Günther, C.W. & van der Aalst W.M.P. (2007). Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso G., Dadam P., Rosemann M. (eds) Business Process Management. BPM 2007. Lecture Notes in Computer Science, vol 4714. Springer, Berlin, Heidelberg

[9] Gaber, M. M., Zaslavsky, A. & Krishnaswamy, S. (2005). Mining Data Streams: A Review. ACM SIGMOD Record, vol. 34 (2), pp. 18-26. USA, New York: ACM.

[10] Widmer, G. & Kubat, M. Mach Learn (1996) 23: 69. doi:10.1007/BF00116900

[11] Manku, G. S., Motwani, R. (2002). Approximate frequency counts over data streams. VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases, pp. 346-357.

[12] Redlich, D., Molka, T., Gilani, W., Blair, G. & Rashid, A. (2014). Scalable Dynamic Business Process Discovery with the Constructs Competition Miner. 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA), vol. 1293 CEUR Workshop Proceedings (pp. 91-107).

[13] Leemans, S.J.J., Fahland, D. & & van der Aalst W.M.P. (2014). Process and Deviation Exploration with Inductive Visual Miner. Eindhoven University of Technology, Netherlands.

[14] Arlitt, M., Cherkasova, L., Dilley, J., Friedriech, R. & Jin, T. (2000). Evaluating content management techniques for Web proxy caches. ACM SIGMETRICS Performance Evaluation Review, vol. 27(4), pp. 3-11.

[15] Romano, S. & el Aarag, H. (2008). A Quantitative Study of Recency and Frequency based Web Cache Replacement Strategies. Proceedings of the 11th communications and networking simulation symposium, pp. 70-78.

Appendix

I. License

Non-exclusive licence to reproduce thesis and make thesis public