UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Jevgeni Martjušev

Efficient Algorithms for Discovering Concept Drift in Business Processes.

Master's Thesis (30 ECTS)

Supervisors:
R.P. Jagadeesh Chandra Bose, Technical University of Eindhoven, Netherlands
Fabrizio Maria Maggi, University of Tartu, Estonia

Author:	…………………………..…	"…..." May 2013
Supervisor:	……………………………..	"…..." May 2013
Supervisor:	……………………………..	"…..." May 2013
Professor:	……………………………..	"…..." May 2013

TARTU 2013

Acknowledgments

This research was supported by national scholarship program Kristjan Jaak, which is funded and managed by Archimedes Foundation in collaboration with the Ministry of Education and Research.

Contents

1. Introduction

Background

A business process is a partially ordered set of activities, which need to be executed to achieve a certain goal. Processes are often supported by information systems that record event logs during the execution of this process. Techniques in the area of process mining analyze event logs to derive information about the process, such as performance bottlenecks, social interactions between involved parties, conditions that alter the flow of the process.

Problem Statement

Process mining algorithms assume that targeted processes are consistent. But in reality, processes can change for many reasons: economic factors (e.g. crisis), adoption of a new law, such as Sarbanes-Oxley Act [1], seasonal peculiarities, deadline escalations [2]. These changes—also known as concept drifts—are listed as one of the main challenges in process mining [3]. In a complex task of analyzing the execution behavior of processes manifested as event logs, we need to discover any concept drift before proceeding with building a process model or applying other process intelligence algorithms. Surprisingly little research was done in the field of concept drift detection in process mining.

Objective

The objective of this thesis is to develop efficient algorithms for dealing with concept drifts. These include: (i) an approach to automatically identify the points (time periods) of change, (ii) a technique for change detection using adaptive windows, (iii) techniques for detecting gradual drifts and multi-order dynamics. All of the algorithms are to be implemented as the Concept Drift plug-in for ProM [4]. Quality of the algorithms are to be assessed using an objective evaluation framework both on simulated and real-life event logs. For generating the simulated logs, we shall model the process using Colored Petri Nets and simulate it in CPN Tools [5].

Document Organization

The rest of this thesis is organized as follows: Chapter 2 covers the preliminaries on the topics of process mining, concept drift, Colored Petri Nets and two software tools – CPN Tools and ProM. Chapter 3 describes the related work in the field of dealing with concept drifts in process mining. Chapter 4 describes four novel approaches for dealing with concept drifts. Chapter 5 contains the results of applying these approaches on both simulated and real-life data. Finally, Chapter 6 discusses the results of this thesis and gives an overview on future work.

2. Preliminaries

Process Mining

Process mining is a business process management technique that acts as a bridge between business process analysis and data mining. There are three key topics in this area: *process discovery*, *conformance checking* and *enhancement*. Process discovery algorithms analyze an event log of a process and output a process model using modeling notations, such as Petri Nets or Event-driven Process Chains. Conformance checking deals with how good a log conforms to an existing process model (acquired from domain experts, for example). Using these techniques one can detect deviations, such as fraudulent executions. Enhancement of a model stands for extending an existing model with additional information from the event log, for example performance data. This allows us to localize performance bottlenecks, recommend process redesign and more.

For further information refer to the “Process Mining” book [6].

Abstractions in Process Mining

An *event log* is a piece of data, recorded during the execution of a single business process during some period of time. Example: an event log of the medical insurance claim process recorded in year 2012 by the information system of an insurance company. A *process instance, case* or *trace* refers to a single execution of the process. Example: John Doe claiming insurance after falling ill during his journey in July 2012. An *activity* or *task* refers to separate actions, executed during the process. Example: during the insurance claim process, “Register” task is executed to insert the data about the claim into the system and "Medical check" task is executed to assess the medical condition of the claimant. An *event* refers to execution of an activity during a single case. Example: John Doe is being assessed during the “Medical Check”. A case is an ordered list of events. Events can have additional *attributes*, such as *timestamps*, *resources*, *cost*, etc. Example: John Doe was examined by doctor Brown (resource) at 16:20:00 on 01.08.2012 (timestamp) and she concluded that Doe's statements about his illness are true (custom boolean attribute).

Concept Drift

Concept drift occurs when something is changed while being analyzed. In process mining we analyze the process taking its event log as input. If the process that generated this log was somehow modified once, then we would like to divide the log into two parts – one before and one after the change was introduced. If the process has changed *n* times, then we can divide the log into *n+1* parts. Almost always it is not explicitly stated when the process has changed. There is a need for algorithms that can discover such changes. If we try to build a process model out of the entire log, which incorporates drifts, then most likely we would see some kind of a combination of two or more processes, which will not be a suitable representation of either processes.

Topics in Dealing with Concept Drift

Three main problems can be identified in dealing with concept drifts in process mining:

1. *Change Detection:* The most important problem is to detect that at some point there was a change.
2. *Change Localization:* After a change is detected we need to identify which part of the process was changed and how.
3. *Unravel Process Evolution:* After changes have been detected and localized, we need to put them into perspective, i.e., describe why and how the process changes.

This thesis focuses on change detection.

Change Perspective

Processes can change in different ways. Three important change perspectives can be defined:

1. *Control-flow Perspective:* change in the structure of the model, such as inserting/deleting/substituting/reordering of activities, splits and joins. Example (depicted using YAWL [7]): three activities were modeled using an OR-split, meaning that either one, two or all of them can be executed (as in Illustration 1), but after a change, the activities are modeled using an XOR-split, meaning that one

and only one of them can be executed (as in Illustration 2). As a result, before the drift there were fifteen possible trace fragments:

- Prepare Notification – By Phone
- Prepare Notification – By Email
- Prepare Notification – By Post
- Prepare Notification – By Phone – By Email
- Prepare Notification – By Phone – By Post
- Prepare Notification – By Email – By Phone
- Prepare Notification – By Email – By Post
- Prepare Notification – By Post – By Phone
- Prepare Notification – By Post – By Email
- Prepare Notification – By Phone – By Email – By Post
- Prepare Notification – By Phone – By Post – By Email
- Prepare Notification – By Email – By Phone – By Post
- Prepare Notification – By Email – By Post – By Phone
- Prepare Notification – By Post – By Phone – By Email
- Prepare Notification – By Post – By Email – By Phone

and after the concept drift only three different trace fragments are possible:

- Prepare Notification – By Phone
- Prepare Notification – By Email
- Prepare Notification – By Post

Notice that three trace fragments can appear both before and after the concept drift. This fact makes change detection more complex, because we can not fully discriminate between traces that are only possible before or after the drift.

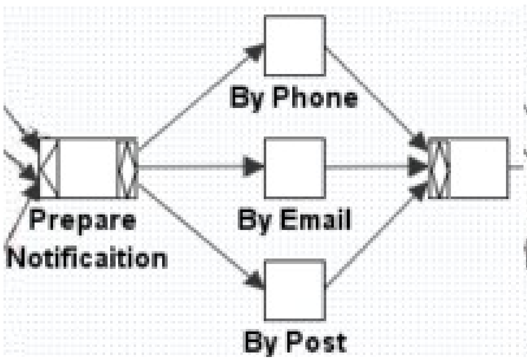


Illustration 1: Part of the process model before concept drift

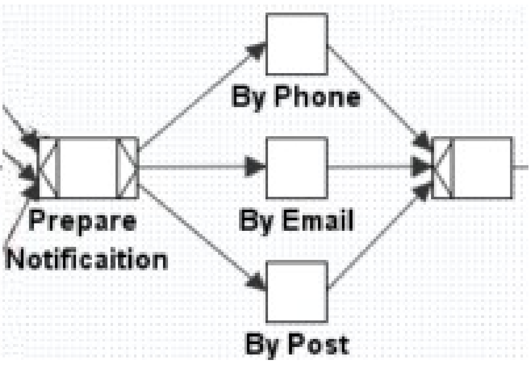


Illustration 2: Part of the process model after concept drift

2. *Data Perspective:* change in the data that goes along with the process, such as changing the way the data is required, used or generated. Example: a new document is required to execute a particular activity.
3. *Resource Perspective:* change in the resources that execute activities, such as changes in roles and organizational structure. Example: a clerk can now execute an activity that was previously a manager's job.

With a single concept drift, the process can change from more than one perspective. This thesis focuses on detecting changes from the control-flow perspective.

Nature of Drifts

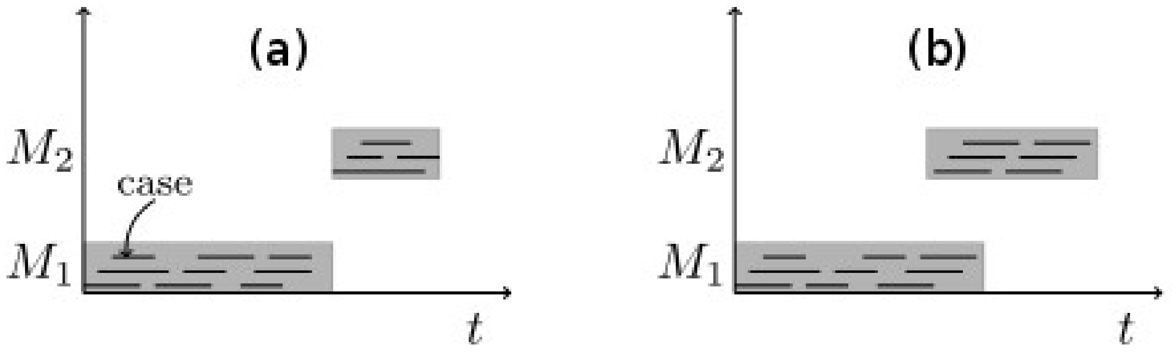
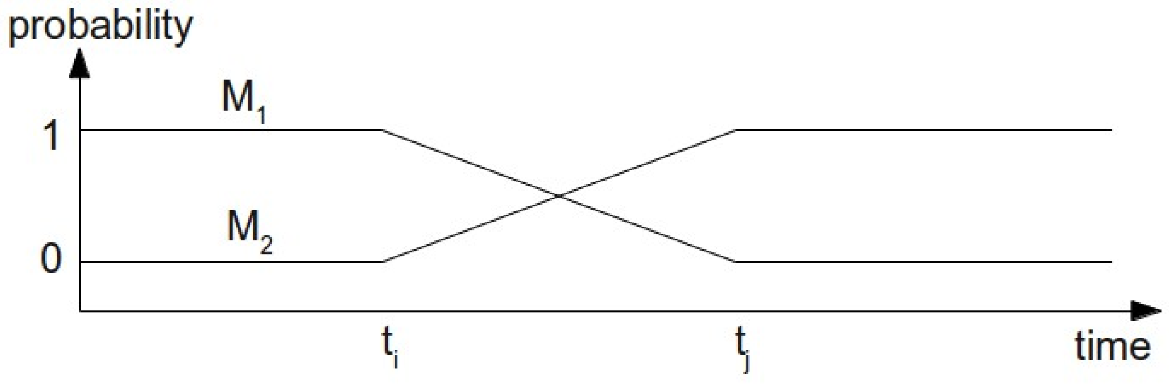


Illustration 3: Distribution of cases for different process variants in case of sudden drift (a) and gradual drift (b) (from [8])

Concept drifts can have different natures. Assume that there is a concept drift in our process, so that there are two modifications of the process: M_1 and M_2. In case of a sudden (or abrupt) drift scenario there is a time instant, so that every trace before it was of type M_1, and after that every trace is of type M_2. In case of a gradual drift scenario, there is a period in time, when both M_1 and M_2 can co-occur. Both scenarios can be seen on Illustration 3. Gradual drifts can have different distributions of process instances.

In case of a *linear graduality* as in Illustration 4 the probability of starting an old process instance (M_1) is 1 before time t_i (which is the start of the drift period). Then the probability linearly decreases until t_j (which is the end of the drift period), after which it stays at 0. This is characterized by function $P(M_1)=\frac{t_j-t}{t_j-t_i}$. At every point in time, the probability of starting the new process type (M_2) is $P(M_2)=1-P(M_1)$.



In case of an *exponential graduality* as in Illustration 5 the probability of starting an old process instance (M_1) is 1 before time t_i (which is the start of the drift period). Then the probability decreases as an exponential decay, characterized by function $P(M_1)=e^{-\lambda(t-t_i)}$ where λ is the decay constant. A bit more intuitive way to describe exponential graduality is by using the notion of half-life $t_{1/2}=\frac{\ln 2}{\lambda}$. In this case, half-life is the time (expressed as number of traces), during which the probability decreases 50%. The function for probability becomes $P(M_1)=(\frac{1}{2})^{(t-t_i)/t_{1/2}}$. Note that there is no fixed end of the drift period, because negative exponent never reaches zero. However, after a long enough time period, the probability of getting old process type would be negligible, so that the drift period can be considered as finished. At every point in time, the probability of starting the new process type (M_2) is $P(M_2)=1-P(M_1)$.

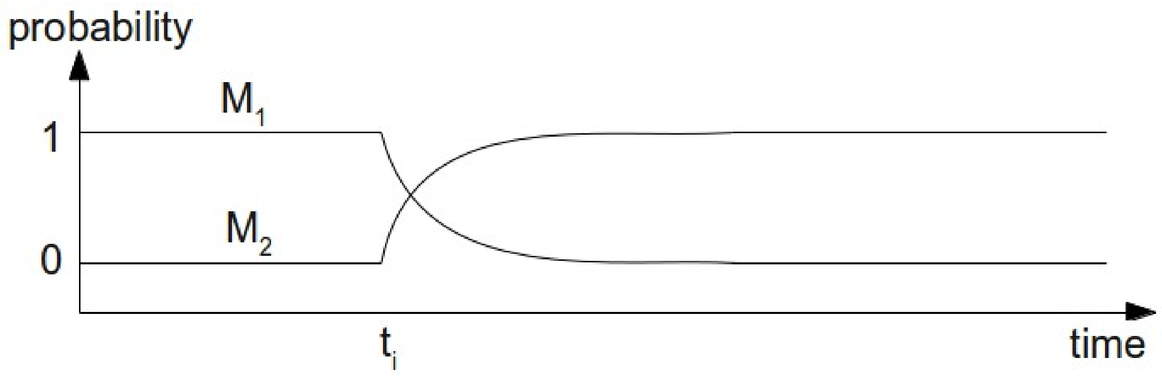
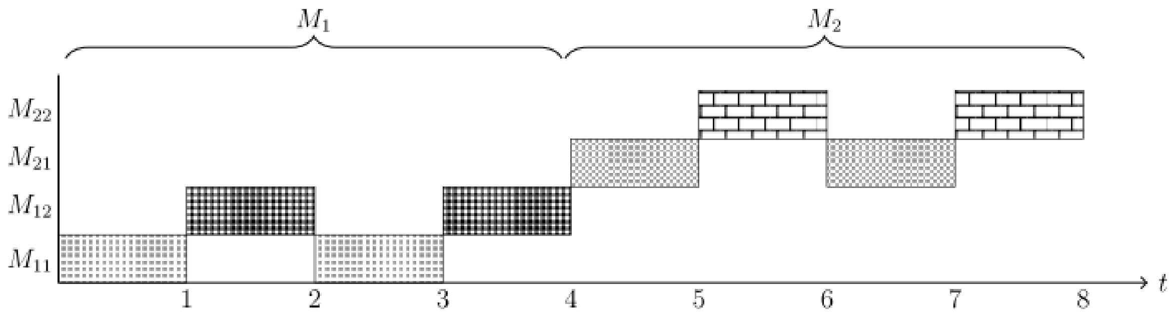


Illustration 5: Probability of starting old and new process variants in case of exponential graduality

Multi-order Dynamics

Every process involves changing something from one state to another – it has a dynamic nature. Concept drift – the process of changing the process - is sometimes referred as second-order dynamics [8]. However, there can be further orders of dynamics. This phenomenon is called multi-order dynamics. This occurs when the process fluctuates at two (or more) different levels, e.g. as on Illustration 6. If you take one time unit as a week, then you will have a recurring minor change every week (change from M_{11} to M_{12} and back, change from M_{12} to M_{22} and back) and a probably recurring major change every 4 weeks (change from M_1 to M_2).



Online vs Offline Detection

Algorithms that aim at detecting concept drifts can be divided into two groups. Offline algorithms assume that the whole process log is available for analysis, while online algorithms are able to work with an open log stream, where events are logged in real-time. The main advantage of online algorithms is that they can be integrated into Process-Aware

Information Systems to detect concept drifts as soon as the process has changed.

Petri Nets

In this thesis, we use Petri Nets as a modeling formalism for representing processes and for generating event logs through simulation. In the following sections, we give a brief introduction to Petri Nets, Colored Petri Nets (an extension of Petri Nets) and CPN Tools (a tool for modeling and simulating Petri Nets).

Petri Nets are a formal and graphical appealing language which is appropriate for modeling systems with concurrency and resource sharing [9]. The classical Petri net is a directed bipartite graph with two types of nodes called *places* and *transitions*. The nodes are connected via directed arcs. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles [10]. Places can contain *tokens*, represented as black dots. A transition may *fire* if every place with an arc to this transition (i.e., input place) has a token. After firing, those tokens are consumed and every place with an arc from this transition (i.e., output place) acquires a token. In case of process modeling, firing a transition is associated with executing an action. There are many extensions that add features to classical Petri Nets.

Colored Petri Nets

Colored Petri Nets (CPN) is a language for the modeling and validation of systems in which concurrency, communication, and synchronization play a major role [11]. It is an extension of Petri Nets modeling language with possibility to associate data ("color") to tokens, track the time of process execution and add custom expressions to arcs and transitions.

CPN Tools

CPN Tools is a tool for editing, simulating, and analyzing Colored Petri Nets [5]. It uses an extension of Standard ML programming language [12] for manipulating token-associated data and coding arc and transition expressions. CPN Tools supports hierarchies for decomposing a model into submodels. Illustration 7 shows CPN Tools with a model opened in it. A green circle with an integer represents the number of tokens in the place. All the place's tokens are listed in a green rectangle. A green border around the "Send

Packet" transition means that it can be fired. The text near places (such as *INTxDATA*) describes the type of token this place accepts. Variables and functions near arcs (such as *(n,p)* or *imin(n1,n2)*) are arc expressions that describe the data, transferred between places, when a transition is fired. In this thesis, we use CPN Tools for modeling and simulating processes to evaluate proposed techniques.

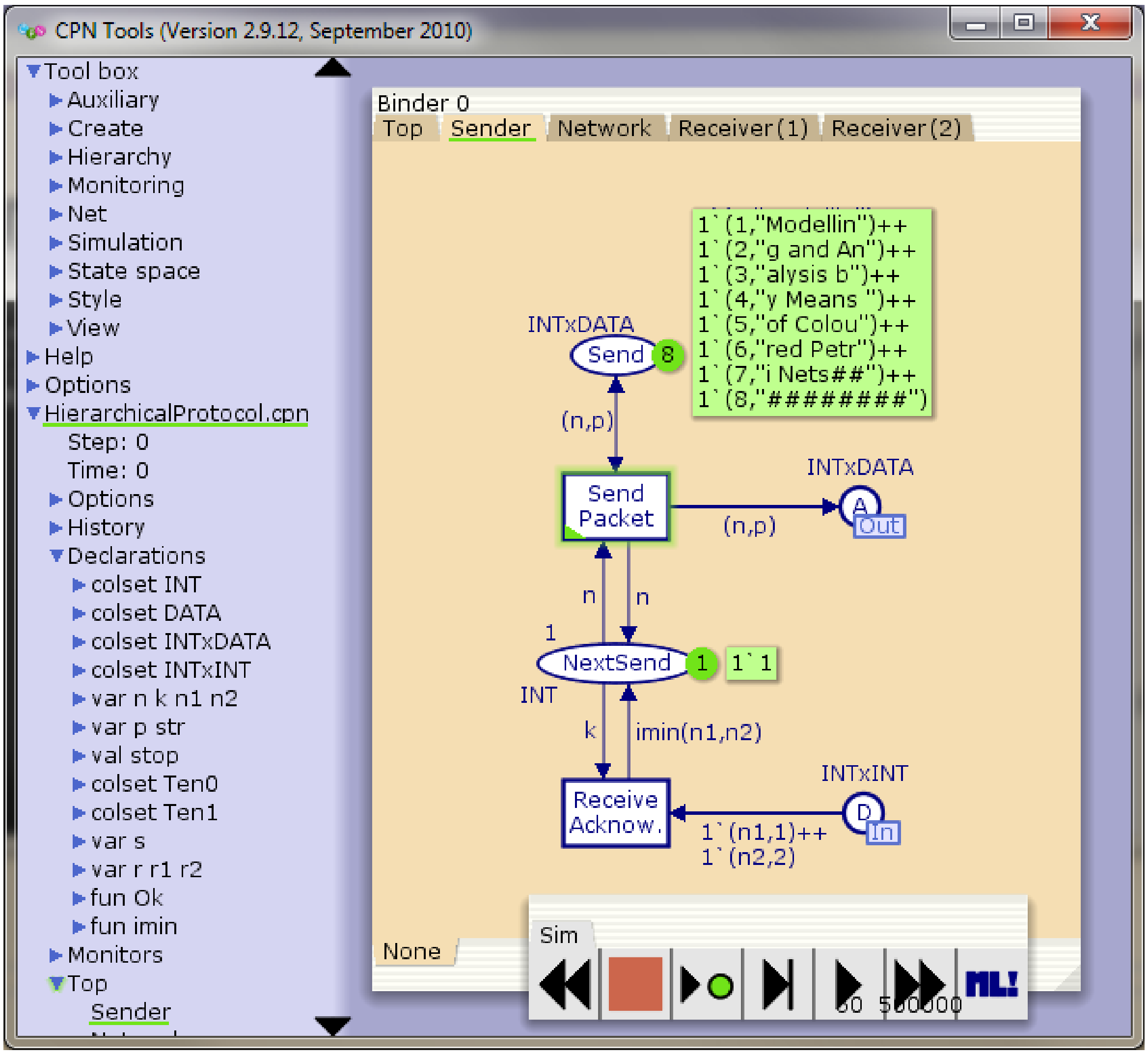


Illustration 7: CPN Tools screenshot from official website with a main menu on the left, model view in the center and a toolbar to control simulation at the bottom.

ProM Framework

ProM is a generic open-source framework for implementing process mining tools in a standard environment [4]. It is written in Java and thus it is platform-independent. There are more than 400 plug-ins available. These allow us to preprocess and import logs, discover process models, do performance analysis etc. In this thesis all the algorithms have been implemented as part of the Concept Drift plug-in. Illustration 8 shows ProM Actions view.

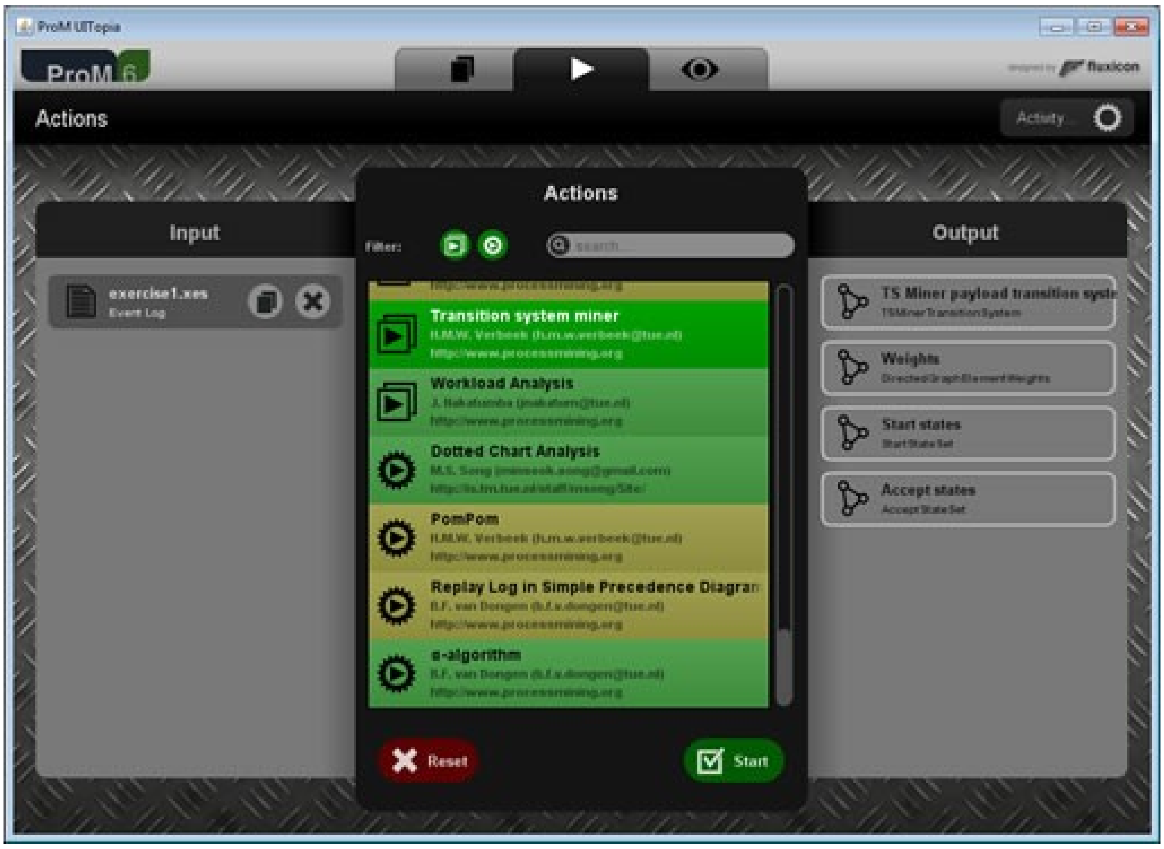


Illustration 8: ProM screenshot from official website with possible actions in the center (Transition system miner is selected), inputs, required for the selected action on the left (an event log is needed) and outputs, produced by the selected action on the right (payload transition system, weights etc).

3. Related Work

Concept Drift in Data Mining

Concept drift is well-studied in the area of data mining and machine learning [13][14][15]. Bifet et al. [16] have implemented a way to learn from data while dealing with concept drifts. Variables are predicted based on the past values, that are contained in an adaptive window (ADWIN) that grows (i.e., encompasses more values) when there is no concept drift and shrinks otherwise. They also provide rigorous guarantees on the ability to detect concept drifts.

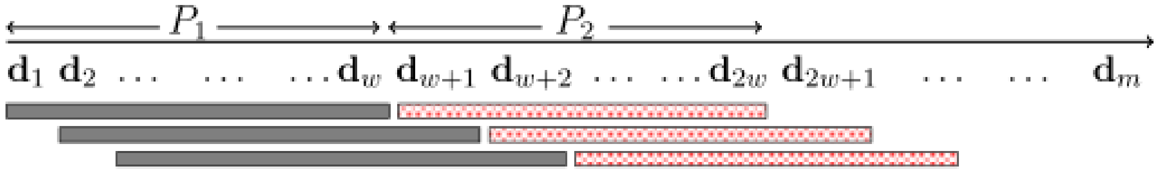
Overview of Changes

Several papers describe a business process can changes [17][18][19]. Günther et al. [20] attempt to describe process changes using process mining techniques. However, this approach assumes that the changes to the system are logged explicitly.

Discovering Concept Drifts using Hypothesis Testing

The most significant work was published by Bose et al. [8] and implemented as the Concept Drift plug-in in ProM. The proposed algorithm relies on the fact, that the characteristics of traces before the concept drift differ from their characteristics after the concept drift. We take the event log, which is basically a bag (multiset) of traces, ordered by the time of the first event in the trace and compute a feature vector d_i for each trace t_i or each sublog, where sublog is a list of consecutive traces. Features defined per sublog are called *global*, while those defined per single trace are called *local*. The features proposed in [8] include two global features: Relation Type Count and Relation Entropy as well as two local features: Window Count and J-Measure. Given a window size l, the window count of an activity pair a,b in a trace t with respect to follows (precedes) relation is the size of the multiset of subsequences in t, where b follows (precedes) a within a window of length l. J-measure is calculated based on the window count and measures the difference between the probability of b to occur/not occur and the probability of b to occur/not occur given that a occurred. The feature can be calculated over every possible activity pair or over chosen pairs.

The algorithm starts by initializing two populations $P1=[d_1 \dots d_w]$ and $P2=[d_{w+1} \dots d_{2w}]$ (where d_i is the vector of feature values, calculated over each chosen activity pair for the trace t_i) of predefined size w and performing a statistical hypothesis test (such as the Kolmogorov-Smirnov test and Mann-Whitney test) for each of the activity pairs. The p-value returned by a test is plotted against w, which is the last trace of the left population. The probability value (p-value) of a statistical hypothesis test is the probability of getting a value of the test statistic as extreme as or more extreme than that observed by chance alone, if the null hypothesis H_0 is true [21]. The null hypothesis in our case states that both populations are from the same distribution. A p-value close to 1 means that most likely there is no drift, while a p-value close to 0 indicates that most certainly there was a change near w. Then we shift the populations using a sliding window by one trace to the right, obtaining a new population pair including $P1=[d_2 \dots d_{w+1}]$ and $P2=[d_{w+2} \dots d_{2w+1}]$, repeat the hypothesis test and plot the p-value (refer to Illustration 9). We shift the populations to the right and plot the p-value until the end the log has been reached.


P_1
P_2
$\mathbf{d}_1$ $\mathbf{d}_2$ ... ...$\mathbf{d}_w$ $\mathbf{d}_{w+1}$ $\mathbf{d}_{w+2}$ ... ...$\mathbf{d}_{2w}$ $\mathbf{d}_{2w+1}$ ... ... $\mathbf{d}_m$

A possible resulting plot is depicted in Illustration 10 with trace indices on x-axis and p-values on y-axis. The four big troughs indicate concept drifts at points 1200, 2400, 3600, 4800. One must manually inspect the plot to detect concept drifts.

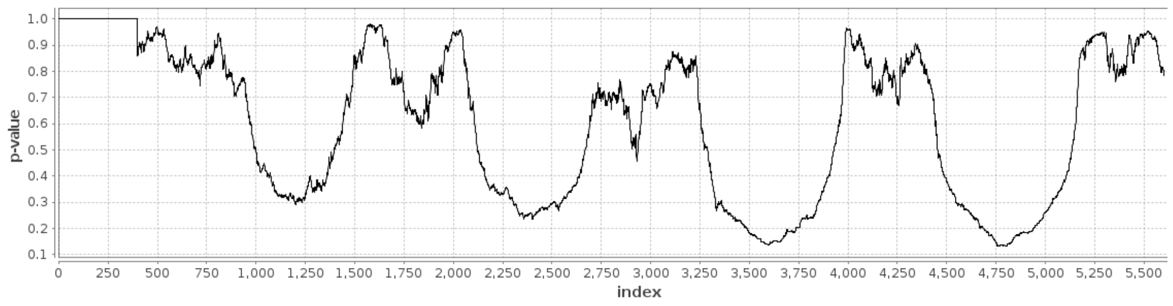


Illustration 10: Resulting plot of p-values against trace numbers produced by Concept Drift plug-in.

The possibility of calculating features over manually selected pairs provides the basic solution for the second problem of dealing with concept drifts – change localization.

Example: by choosing activities *a* and *b* as the first ones and *c* as the second one, we can calculate the probability value for only two activity pairs: *(a,c)* and *(b,c)*, thus acquiring information about whether the process changed with respect to those two pairs. This approach does not give the complete answer about how exactly the process changed.

The approaches proposed in this thesis highly rely on the ideas proposed in [8]. We shall further refer to this algorithm as the "original algorithm".

Discovering Concept Drifts using Abstract Representation

Carmona and Gavaldà [22] proposed an online algorithm for detecting concept drifts. Every prefix of every trace is converted into a Parikh vector, defined as $\hat{\sigma} = (\#(\sigma, t_1), \#(\sigma, t_2), \ldots, \#(\sigma, t_n))$ for the trace $\sigma \in \{t_1, t_2, \ldots, t_n\}^*$. They build an abstract representation (a polyhedron) from Parikh vectors on random trace prefixes by calculating the convex hull of those vector points. Next, Parikh vectors of subsequent traces are tested on whether they lie within the polyhedron. When a sufficient amount of samples lie outside of the polyhedron – a concept drift is declared. The exact trace number of the drift is not reported. Results are evaluated by the number of sampled points it takes to detect the drift. This algorithm is only suitable for detecting sudden drifts without multi-order dynamics. The current implementation of the algorithm has some flaws: it terminates as soon as it detects a concept drift, thus it can not detect more than one drift.

Clustering-based Concept Drift

Luengo and Sepúlveda [23] use clustering for detecting concept drifts. Every trace is converted to a vector of *maximal repeats*. As defined by Bose [24], a *maximal repeat* in a sequence **s** is defined as a subsequence α that occurs in a *maximal pair* in **s**. A *maximal pair* in a sequence **s** is a subsequence α that manifests in **s** at two distinct positions *i* and *j* such that the element to the immediate left (right) of the manifestation of α at position *i* is different from the element to the left (right) of the manifestation of α at position *j*. The time dimension is further added to each vector. Agglomerative Hierarchical Clustering (AHC) with the minimum variance criterion is used as the clustering algorithm. There is no publicly available implementation of the algorithm to test or integrate. Three fairly simple synthetic examples were used to test the algorithm; the measured metric was the accuracy of clustering. Their algorithm had assigned the traces to correct clusters in 70% to 100% cases.

Discovering Concept Drifts by Comparing Mined Models

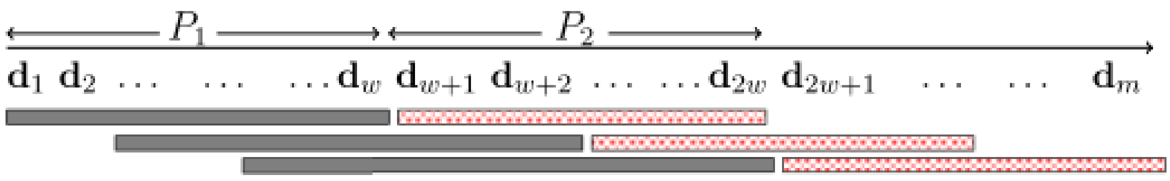
Weber et al. [25] propose another online approach. They repeatedly mine models from sublogs and use hypothesis testing to make sure that the process model or its probabilistic deterministic finite automata (PDFA) representation has changed significantly from the ground truth. Here, the challenge is to choose a sufficient amount of traces to mine a representative model.

4. Approaches

All the approaches proposed in this thesis are further developments of the algorithms proposed by Bose [8], which are implemented in the Concept Drift plug-in of ProM. These approaches were already briefly presented in [26]. All of them are implemented as offline algorithms, i.e., you need to import an entire event log into ProM before the analysis can start. But every algorithm can be slightly modified to be used in an online setting.

Step Size Improvement

In the original algorithm [8], after each hypothesis test we have shifted the populations one trace at a time, e.g. if we compared traces *1* to *i* with traces *i+1* to *2i,* then at the next step we would have to compare traces *2* to *i+1* with traces *i+2* to *2i+1*. This is computationally expensive, because the number of hypothesis tests is directly proportional to the number of traces in the event log. One can improve this by shifting the population *k* traces at a time as in Illustration 11, where the step size (*k*) is *2* traces. Note the difference between this variant, and the one in Illustration 9. For the resulting p-value plot to be continuous, we extend the computed value to the *k* next traces, i.e., if p-value at point *i* is *p(i)* and the next computed value is at point *i+k*, then *p(i+1)=p(i+2)=...=p(i+k-1)=p(i)*. Theoretically, the execution time should reduce k times, but if we choose too large *k*, then we might lose some significant information. There is currently no algorithm for selecting the best value



P_1
P_2
$\mathbf{d}_1\ \mathbf{d}_2\ \ldots\ \ldots\ \ldots\mathbf{d}_w\ \mathbf{d}_{w+1}\ \mathbf{d}_{w+2}\ \ldots\ \ldots\mathbf{d}_{2w}\ \mathbf{d}_{2w+1}\ \ldots\ \ldots\ \mathbf{d}_m$

Illustration 11: Three population pairs of size w with a step size of 2

for *k*. A rule of thumb is that you choose an initial value for the step size, e.g. *k=20*. Then, if the resulting plot looks not very detailed, then decrease *k* and if the algorithm takes too long, then increase *k* for next runs. For examples refer to Computational complexity evaluation section. The step size improvement can be applied to the original algorithm and every algorithm, proposed in this thesis.

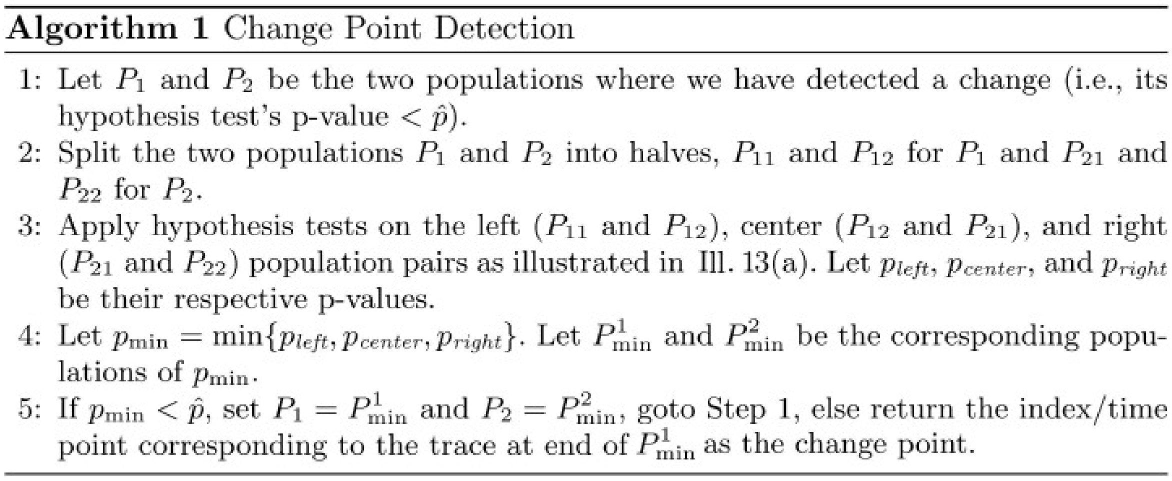
Automatic Change Point Detection

As shown in Illustration 10, the plot needs to be manually inspected for changes. Intuitively, there are 4 changes, at points 1200, 2400, 3600 and 4800, though the troughs at 1850 and 2900 could also be considered significant. Automatic detection could be important for automated process mining tools, which do not wait for inputs from the user, other than the event log itself. It would be beneficial for this type of programs, to be able to run the concept drift detection algorithm, extract the change points and then build *n+1* models instead of one (in case *n* drifts were detected).

One could think of a naïve algorithm for automatic change detection, where the drift point is denoted as the center of the plot region, where the p-value is less than some chosen threshold. This approach has a major flaw in accuracy. Refer to Illustration 10. If the threshold is set to 0.3, then this algorithm would find 7 drifts instead of 4 because of the fluctuations near the 0.3. For example, notice that there is a bump upwards near the 2520. This would split the region below 0.3 into two, thus returning two change points.

The algorithm proposed in this thesis does not have this flaw. Pseudocode from the original publication [26] is shown in Illustration 12. We first define some p-value threshold $\hat{p}$. When we detect a p-value less than it, we start the change point detection algorithm. It tries to find the closest point using recursive bisection, as depicted in Illustration 13. Example: assume that we have chosen p-value threshold $\hat{p}=0.3$ and we were comparing $P_1=[d_1 \ldots d_w]$ and $P_2=[d_{w+1} \ldots d_{2w}]$ when we first detected a p-value less than 0.3 (step 1). This started the change point detection algorithm. Now we split those population into four new ones:

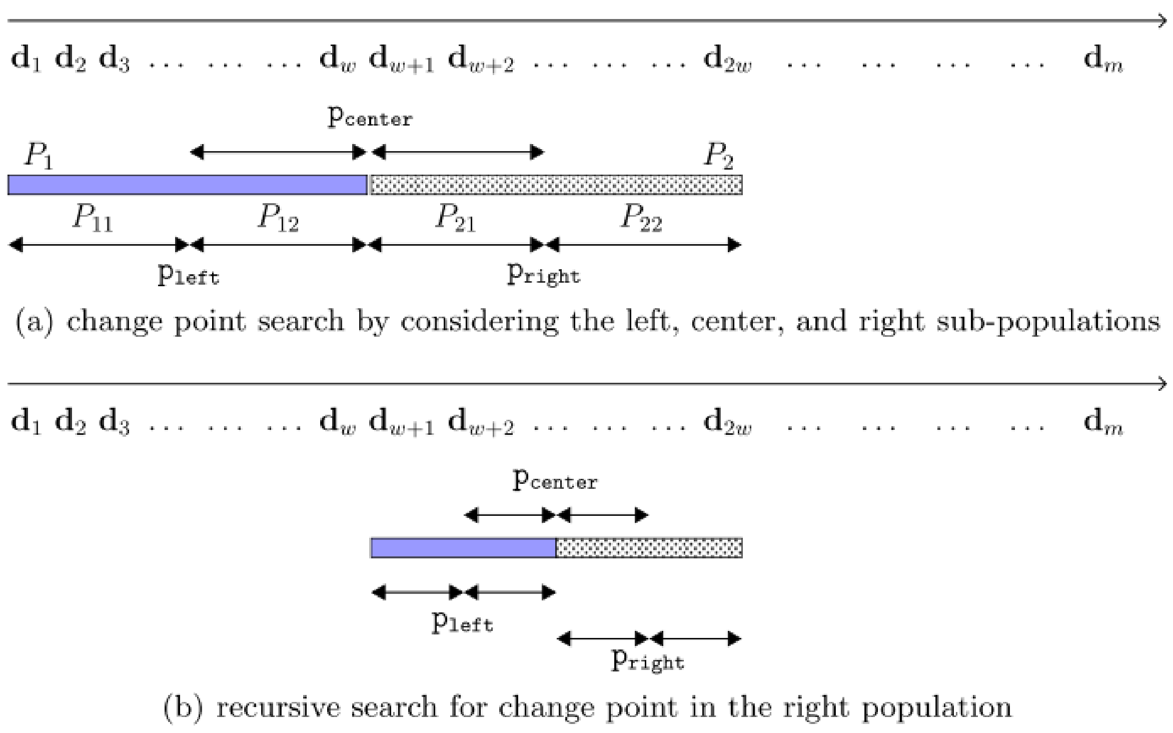
$P_{11}=[d_1 \dots d_{w/2}]$, $P_{12}=[d_{w/2+1} \dots d_w]$, $P_{21}=[d_{w+1} \dots d_{3w/2}]$ and $P_{22}=[d_{w+1} \dots d_{2w}]$ (step 2). We apply three hypothesis tests, thus acquiring three p-values: p_{left} from P_{11} and P_{12}, p_{center} from P_{12} and P_{21}, p_{right} from P_{21} and P_{22} (step 3, Illustration 13(a)). Assume that $p_{right}=0.2$ was the lowest of three, thus we assume that the concept drift is located inside P_2. The new population pair includes $P^1_{min}=P_{21}$ and $P^2_{min}=P_{22}$ (step 4). $p_{right}=0.2<\hat{p}=0.3$, thus we set $P_1=P^1_{min}=[d_{w+1} \dots d_{3w/2}]$ and $P_2=P^2_{min}=[d_{3w/2+1} \dots d_{2w}]$ (step 5) and continue the algorithm recursively (step 1). Once again we split two populations into four: $P_{11}=[d_{w+1} \dots d_{5w/4}]$, $P_{12}=[d_{5w/4+1} \dots d_{3w/2}]$, $P_{21}=[d_{3w/2+1} \dots d_{7w/4}]$ and $P_{22}=[d_{7w/4+1} \dots d_{2w}]$ (step 2) and apply three hypothesis tests (step 3, Illustration 13(b)). Assume that $p_{center}=0.35$ was the lowest of three, thus we assume that the concept drift is located near the connection of P_1 and P_2. The new population pair includes $P^1_{min}=P_{12}=[d_{5w/4+1} \dots d_{3w/2}]$ and $P^2_{min}=P_{21}=[d_{3w/2+1} \dots d_{7w/4}]$ (step 4). This time $p_{center}=0.35>\hat{p}=0.3$, thus the algorithm stops and reports $d_{3w/2}$ as the change point.



Algorithm 1 Change Point Detection

```
1: Let $P_1$ and $P_2$ be the two populations where we have detected a change (i.e., its
   hypothesis test's p-value $< \hat{p}$).
2: Split the two populations $P_1$ and $P_2$ into halves, $P_{11}$ and $P_{12}$ for $P_1$ and $P_{21}$ and
   $P_{22}$ for $P_2$.
3: Apply hypothesis tests on the left ($P_{11}$ and $P_{12}$), center ($P_{12}$ and $P_{21}$), and right
   ($P_{21}$ and $P_{22}$) population pairs as illustrated in Ill. 13(a). Let $p_{left}$, $p_{center}$, and $p_{right}$
   be their respective p-values.
4: Let $p_{\min} = \min\{p_{left}, p_{center}, p_{right}\}$. Let $P^1_{\min}$ and $P^2_{\min}$ be the corresponding popu-
   lations of $p_{\min}$.
5: If $p_{\min} < \hat{p}$, set $P_1 = P^1_{\min}$ and $P_2 = P^2_{\min}$, goto Step 1, else return the index/time
   point corresponding to the trace at end of $P^1_{\min}$ as the change point.
```

Illustration 12: Change point detection algorithm



$\mathbf{d}_1\ \mathbf{d}_2\ \mathbf{d}_3 \dots \dots \dots \mathbf{d}_w\ \mathbf{d}_{w+1}\ \mathbf{d}_{w+2} \dots \dots \dots \mathbf{d}_{2w} \dots \dots \dots \dots \mathbf{d}_m$
p_{center}
P_1
P_2
P_{11}
P_{12}
P_{21}
P_{22}
p_{left}
p_{right}
(a) change point search by considering the left, center, and right sub-populations
$\mathbf{d}_1\ \mathbf{d}_2\ \mathbf{d}_3 \dots \dots \dots \mathbf{d}_w\ \mathbf{d}_{w+1}\ \mathbf{d}_{w+2} \dots \dots \dots \mathbf{d}_{2w} \dots \dots \dots \dots \mathbf{d}_m$
p_{center}
p_{left}
p_{right}
(b) recursive search for change point in the right population

Illustration 13: Recursive bisection

Adaptive Windows

The algorithm by Bose [21] is highly dependent on the chosen population size *w*. If this parameter is too small, then the plot can contain a lot of noise. As a result, either the noise can be misinterpreted as concept drifts (i.e., we obtain false positives) or concept drifts will be left undetected between random troughs as in Illustration 14, where concept drifts at traces 1200 and 2400 are indistinguishable from the surrounding noise. If the population size becomes too large then the time complexity of the algorithms worsens and some drifts may become undetected (i.e., we obtain false negatives) as in Illustration 15, where concept drift at traces 1200 and 2400 are once again indistinguishable. It also took more than 440 seconds to run the algorithm with such population size.

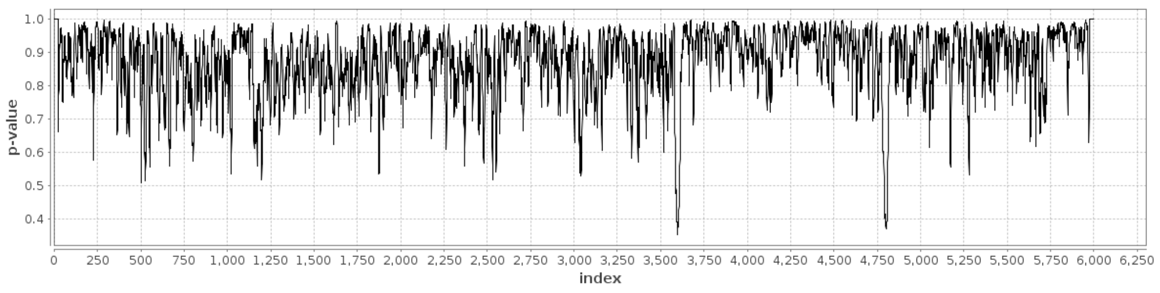


Illustration 14: False positives/false negatives with a small population size of 25

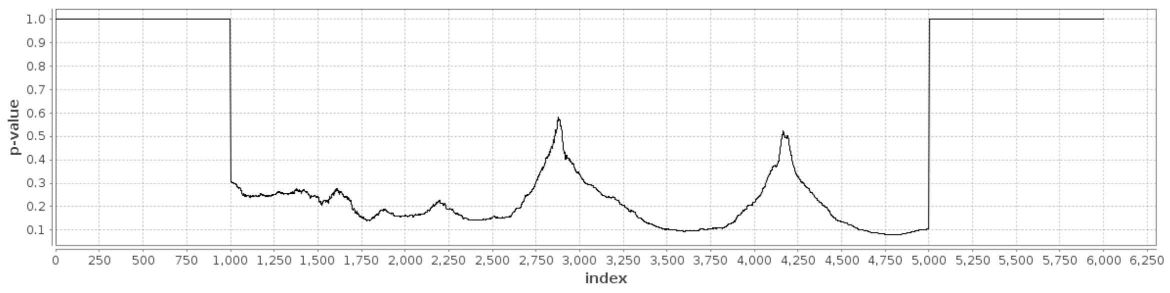


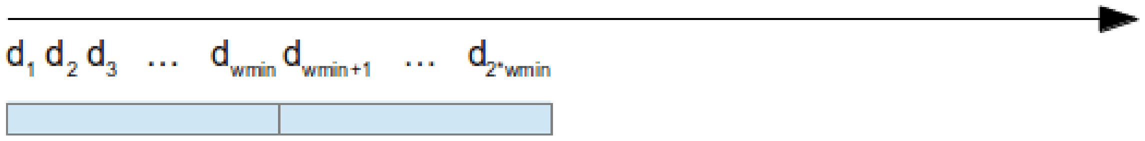
Illustration 15: False negatives with a large population size of 1000

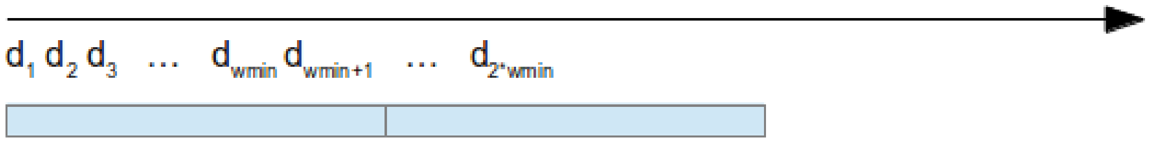
To relax the dependency on the population size, the algorithm is modified by adapting the ADWIN approach [16] to process mining. Pseudocode from the original publication [26] is shown in Illustration 16. We shall further refer to this algorithm as ADWIN.

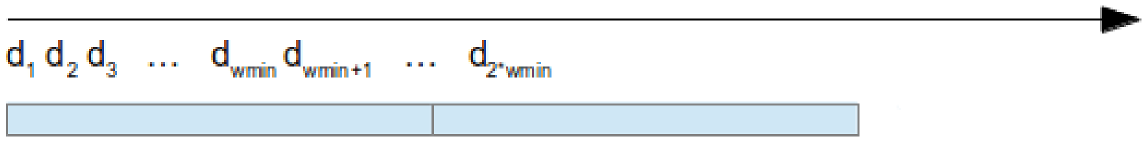
The adjustment of population sizes is depicted in the following illustrations. Let the minimum population size be w_{min}, the maximum population size be w_{max}, and the step size be 1. We first initialize two populations of size w_{min} as in *Illustration 17,* obtaining $P_{left}=[d_1 \dots d_{wmin}]$ and $P_{right}=[d_{wmin+1} \dots d_{2wmin}]$ (step 1). Then we apply the hypothesis test (step 3) and assume that the p-value is greater than threshold ($p>\hat{p}$) (step 7). In this case we extend both populations by 1, which is the step size (step 8). The new populations are $P_{left}=[d_1 \dots d_{wmin+1}]$ and $P_{right}=[d_{wmin+2} \dots d_{2wmin+2}]$ as depicted in *Illustration 18.* Once again we perform a hypothesis test (step 3) resulting in a p-value too high ($p>\hat{p}$) (step 7), and thus we extend both populations by 1 (step 8), obtaining $P_{left}=[d_1 \dots d_{wmin+2}]$ and $P_{right}=[d_{wmin+3} \dots d_{2wmin+4}]$ as in *Illustration 19.* At the same time the size of a single population has reached its maximum (step 9), which can be seen in *Illustration 22,* where the size of P_{right} is $(x+w_{max})-(x+1)+1=w_{max}$. In this case we discard the left population and convert the right one into new population pair (step 9), so that $P_{left}=[d_{x+1} \dots d_{wmax/2}]$ and $P_{right}=[d_{wmax/2+1} \dots d_{wmax}]$ (*Illustration 21*). We perform the hypothesis test (step 3) and the p-value is greater than the threshold (step 7), thus we extend the populations to be $P_{left}=[d_{x+1} \dots d_{wmax/2+1}]$ and $P_{right}=[d_{wmax/2+2} \dots d_{wmax+2}]$ (step 8) (*Illustration 21).* We perform a hypothesis test (step 3) on the new populations and this time we detect a lower p-value ($p<\hat{p}$) (step 4). We start the Change Point Detection algorithm, described in the previous section and assume that it outputs, that a drift was discovered at point d_{drift} (step 5) (Illustration 23). Now we discard the old population pair and initialize new populations of minimum size at $P_{left}=[d_{wmax+3} \ldots d_{wmax+wmin+2}]$ and $P_{right}=[d_{wmax+wmin+3} \ldots d_{wmax+2*wmin+2}]$ (step 6) (Illustration 24). The same procedure is applied until the end of the log has been reached (step 12).

```
Algorithm 2 Change Detection Using Adaptive Windows
Require: a minimum population size $w_{\min}$, a maximum population size $w_{\max}$, p-value
    threshold $\hat{p}$, a step size $k$, and a data stream of values $\mathcal{D}$
 1: Let $P_{\mathtt{left}}$ and $P_{\mathtt{right}}$ be two populations of size $w_{\min}$ with $P_{\mathtt{right}}$ starting at the first
    index after the end of $P_{\mathtt{left}}$.
 2: repeat
 3:     Apply hypothesis test over $P_{\mathtt{left}}$ and $P_{\mathtt{right}}$. Let $p$ be its p-value.
 4:     if $p < \hat{p}$ then
 5:         Identify the change point within $P_{\mathtt{left}}$ and $P_{\mathtt{right}}$ using Algorithm 1.
 6:         Create two new populations $P'_{\mathtt{left}}$ and $P'_{\mathtt{right}}$ of size $w_{\min}$ with $P'_{\mathtt{left}}$ starting
    at the first index after the end of $P_{\mathtt{right}}$ and $P'_{\mathtt{right}}$ starting at the first index after
    the end of $P'_{\mathtt{left}}$. Set $P_{\mathtt{left}} = P'_{\mathtt{left}}$ and $P_{\mathtt{right}} = P'_{\mathtt{right}}$.
 7:     else
 8:         Extend the left and right populations by step size $k$. Reassign the right
    population to start at the first index after the end of the extended left population
    $P_{\mathtt{left}}$.
 9:         if the size of the population $\geq w_{\max}$ then discard the left population $P_{\mathtt{left}}$.
    Split the right population $P_{\mathtt{right}}$ into two halves and use them as the left and right
    populations.
10:         end if
11:     end if
12: until the end of $P_{\mathtt{right}}$ doesn't reach the end of $\mathcal{D}$
```

Illustration 16: Change detection using adaptive windows algorithm


$d_1\ d_2\ d_3\ \dots\ d_{wmin}\ d_{wmin+1}\ \dots\ d_{2*wmin}$
Illustration 17: Initial state (step 1)


$d_1\ d_2\ d_3\ \dots\ d_{wmin}\ d_{wmin+1}\ \dots\ d_{2*wmin}$
Illustration 18: Extension of populations (step 8)


$d_1\ d_2\ d_3\ \dots\ d_{wmin}\ d_{wmin+1}\ \dots\ d_{2*wmin}$
Illustration 19: Another extension (step 8)

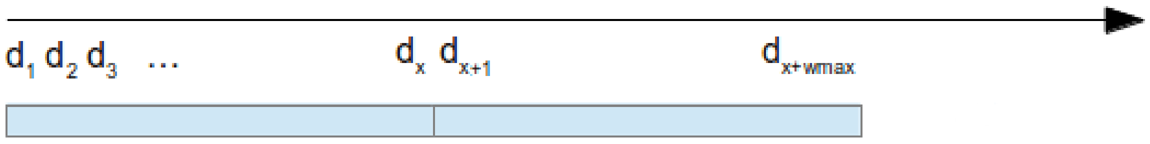


Illustration 20: Size of a population has reached maximum (step 9)

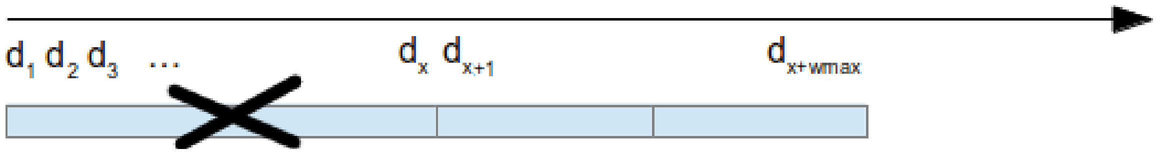


Illustration 21: Discarding the left population (step 9)

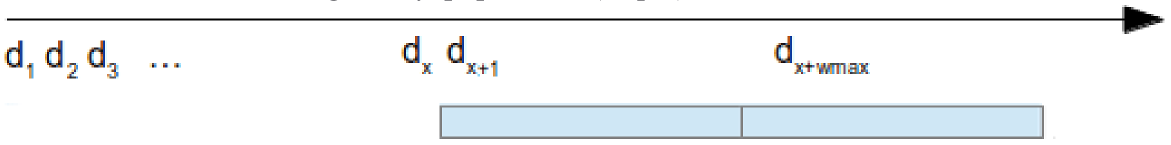


Illustration 22: Continuing the extension (step 8)

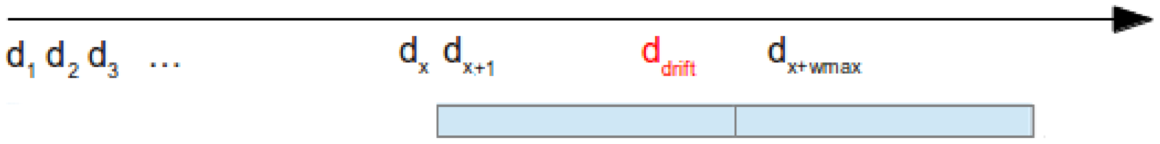


Illustration 23: Drift discovered (step 5)

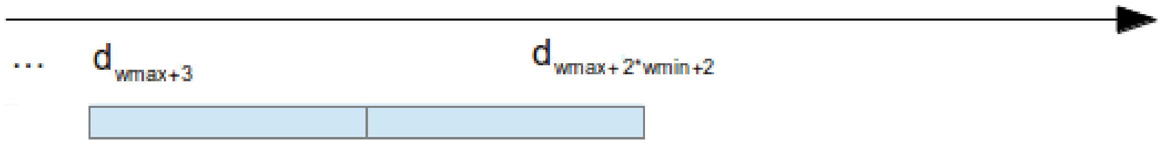


Illustration 24: New populations after drift detection (step 6)

At the end, this algorithm outputs a list of trace numbers, where concept drift was detected and also a p-value plot, similar to the one generated by the original algorithm. The values are plotted against the end of the left population, but there are situations, in which the algorithm discards the populations and creates new ones (steps 6 and 9). This creates a gap between the ends of old and new left populations. For the plot to be continuous, algorithm connects those points with a straight line.

Capturing Gradual Drifts with Non-continuous Populations

Running the algorithm from [8] on a log with gradual drift can offer some positive results, but not very accurate results because of the gradual transition from one process type to another. The hypothesis test's p-value is the lowest when the populations are the most diverse. In case of a gradual drift, during every hypothesis test, there will be instances of both types in both populations, thus p-values will be notably higher, than in the case of a sudden drift. Also it would be even more difficult (than in sudden drift case) to analyze the plot. Refer to Illustration 25, which is the resulting plot of the original algorithm, applied to a log with linear graduality. One can assume that there were some drifts, but it is unclear, how to derive where the start and end points of change period are.

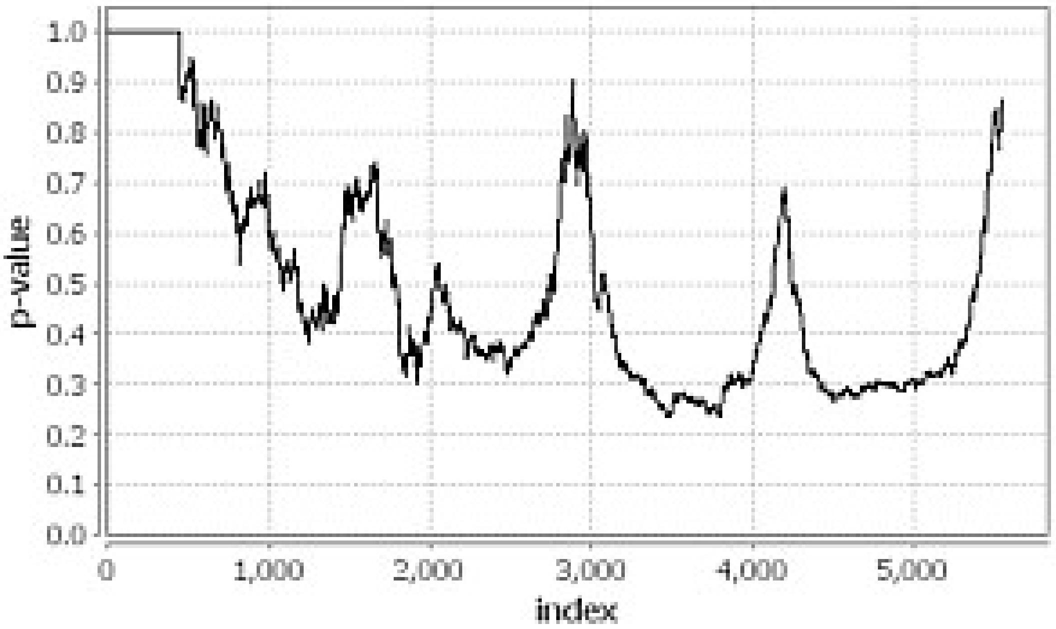


Illustration 25: Original algorithm applied to a log with gradual concept drift

Therefore, the idea for capturing gradual drifts is to use non-continuous populations, i.e., if the last trace of the left population is *I,* then the first trace of the right population shall be not $i+1$, but $i+g$, where $g>1$ is the gap size. The intuition behind this is the following: with a proper gap size, at some step of the algorithm the whole change period shall be contained within the gap, thus P_1 shall contain only traces from M_1 and P_2 shall contain only traces from M_2, where M_1 is the old process type and M_2 is the new process type. This should result in the desired low p-values.

One can modify the original algorithm by introducing non-continuous populations. When the population size is w and gap size is 2, then the initial population pair would include $P_1=[d_1 \dots d_w]$ and $P_2=[d_{w+3} \dots d_{2w+2}]$ and in the next step it would include $P_1=[d_2 \dots d_{w+1}]$ and $P_2=[d_{w+4} \dots d_{2w+3}]$ like in Illustration 26. Note the difference between this variant, and the one in Illustration 9.

Not only the original algorithm, but also ADWIN can be modified to have a gap between populations. In this case, the gap size can be constant, like in the previous example, or varying in size, being equal to the left and right populations. In both cases, at every step of ADWIN, where there is a creation or modification of populations (steps 1,6,8,9 in Illustration 16), the algorithm should insert a gap between the two populations. Also when the gradual ADWIN detects a p-vlaue less than threshold then a gap shrinking algorithm is started. Consider the following modifications to the algorithm (differences from original ADWIN will be underlined), both for the case with a constant size gap of size g and the varying one (the latter will be in parentheses):

1: Let P_{left} and P_{right} be two populations of size w_{min} with P_{right} starting <u>in g (w_{min}) traces</u> after the end of P_{left}.

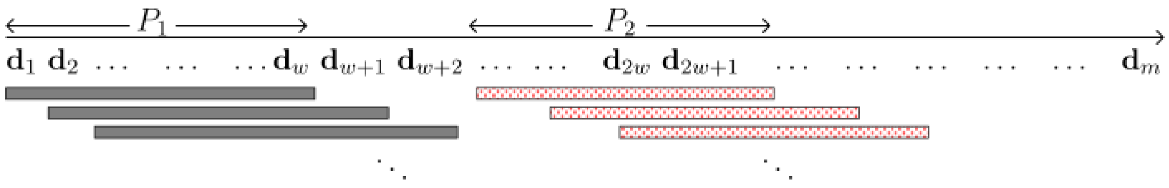
5: Identify change period <u>using gap shrinking algorithm (Illustration 27).</u>

6: Create two new populations P'_{left} and P'_{right} of size w_{min} with P_{right} starting <u>in g (w_{min}) traces</u> after the end of P_{left}. Set $P_{left} = P'_{left}$ and $P_{right} = P'_{right}$.

8: Extend the left and right populations by step size k. Reassign the right population to start <u>in g (w_{min}) traces</u> after the end of the extended left population P_{left}.

9: If the size of the population $\geq w_{max}$ then discard the left population P_{left}.

Split the right population P_{right} into <u>three parts, so that second one is of size g (w_{min}) and first is equal to third. Use first one as P_{left} and third one as P_{right}.</u>



P_1
P_2
$\mathbf{d}_1$ $\mathbf{d}_2$ … … … $\mathbf{d}_w$ $\mathbf{d}_{w+1}$ $\mathbf{d}_{w+2}$ … … $\mathbf{d}_{2w}$ $\mathbf{d}_{2w+1}$ … … … … … $\mathbf{d}_m$

Illustration 26: Three population pairs of size w with a gap of size 2

Algorithm 3 Gap Shrinking

```
1: Let $P_1$ and $P_2$ be the two populations where we have detected a change
   (i.e., its hypothesis test's p-value $p_{original} > \hat{p}$) and $G$ be the gap be-
   tween $P_1$ and $P_2$. Let $k$ be the step size.
2: Let $P_1'$ be the population of the same size as $P_1$, but with start and end
   points shifted $k$ traces to the right.
   Let $P_2'$ be the population of the same size as $P_2$, but with start and end
   points shifted $k$ traces to the left.
3: Let $p_{left}$ be the p-value of hypothesis test applied on $P_1'$ and $P_2$. Let
   $p_{right}$ be the p-value of hypothesis test applied on $P_1$ and $P_2'$.
4: Let $p_{min} = min\{p_{left},\ p_{original},\ p_{right}\}$
5: If $p_{min} = p_{left}$ then set $P_1 = P_1'$ and goto Step 1, else if $p_{min} = p_{right}$
   then set $P_2 = P_2'$ and goto Step 1, else return $G$ as the change period.
```

Illustration 27: Detecting change period of a gradual concept drift using gap shrinking

The intuition behind the gap shrinking algorithm is that we decrease the gap size and do hypothesis tests until the new p-value is greater than old one. This means we have successfully identified the change period.

Capturing Multi-order Dynamics using Time Periods

Existing algorithms do not distinguish between different dynamics orders. They take input parameters, such as population and step size in terms of number of traces. This leads to an inability to define that, for example, we are not interested in analyzing periods shorter than one week. We propose a modification to the existing algorithms, where the input parameters are given in terms of time periods, e.g. minimum population should encompass traces of a period of one week duration. In this variant ADWIN's population minimum size becomes minimum period, population maximum size becomes maximum period and step size becomes step period. Populations with cases from shorter periods can be used to detect micro-level drifts, while those of a larger period can be used to detect macro-level drifts (refer to Illustration 6). This approach can be applied to ADWIN with both continuous and non-continuous populations. A notable difference from the algorithms where population size is given in terms of number of traces is that left and right populations can be different in size, e.g. if the current population size is one day and we are comparing a day, when 50 traces had started (i.e., the left population) to the next day, when 100 traces had started (i.e., the right population).

5. Experimental Setup and Evaluation

Description of the Simulated Process and Means of Generating the Logs

For some of the experiments we shall use the health insurance claim log, used by Bose et al. [8]. This log was already used previously in this thesis to illustrate some of the points, e.g. in Illustration 10. It contains 15 activities, 6000 traces, 58783 events and four sudden concept drift change points at 1200, 2400, 3600 and 4800. We will further refer to this log as the “original insurance claim log”.

For other experiments, we have remodeled the same insurance claim process in CPN Tools and generated the logs with desired concept drift characteristics (graduality/multi-order dynamics). The process of log generation is described below. It could be easily reused by anyone, who is conducting research in the field of concept drift detection and is in need for a method to generate test logs.

Before the release of "KeyValue" plug-in for ProM, the generation of event logs in a suitable format (MXML or XES) during the CPN Tools simulation was quite complex. It involved invoking custom ML functions for every activity in the model to generate MXML log statements [27].

For this thesis we used a newer simpler approach that involves importing CPN Tools simulation logs using the "Convert Key/Value Set to Log" action from the “KeyValue” plug-in [28]. The possibility of choosing different models to mimic the concept drift is implemented using the hierarchy functionality of CPN Tools [29].

1. Create two or more different models on different pages in CPN Tools depicting the process before and after the concept drift. Illustration 28 and Illustration 29 show two slightly different models we used to generate logs with gradual concept drift. More than two models are needed in case logs incorporate more than one non-recurring concept drift.

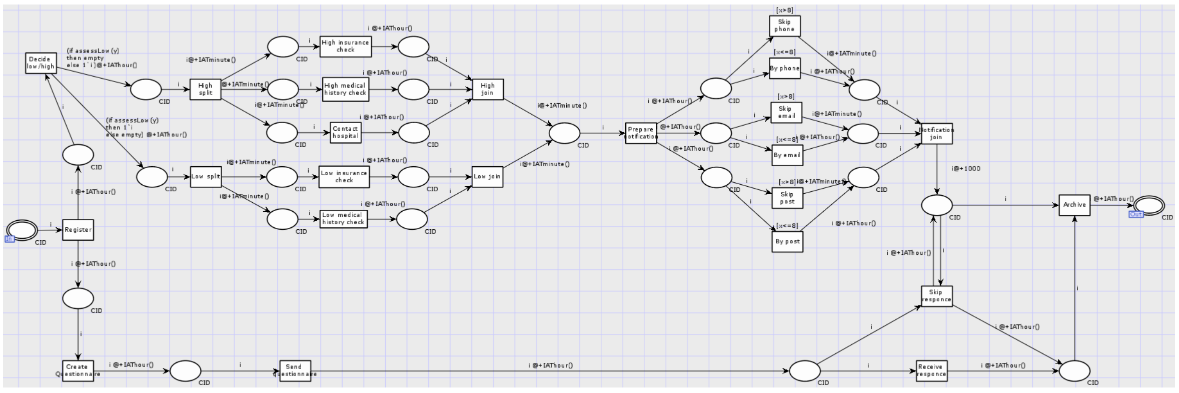


Illustration 28: Submodel 1

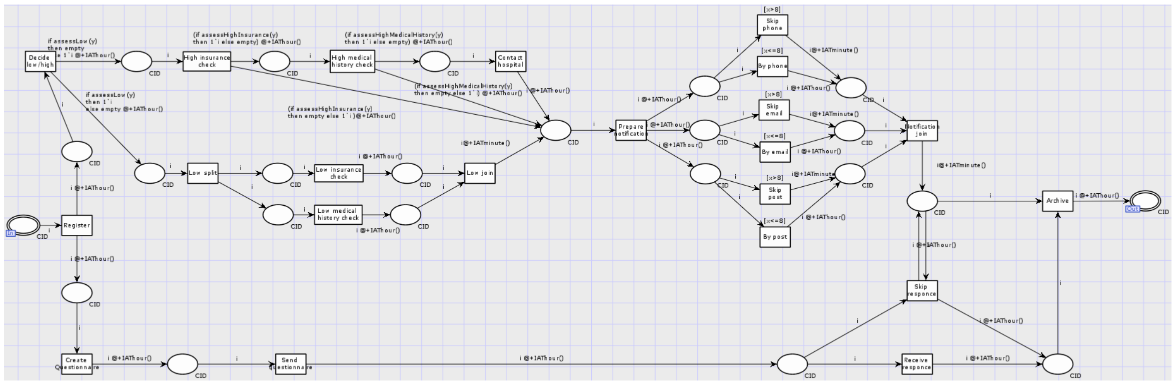


Illustration 29: Submodel 2

2. Create a page with the orchestrating model. An example is depicted in Illustration 30. There are two substitution transitions (Submodel1 and Submodel2). Submodel1 encapsulates the logic of the process variant from Illustration 28 and Submodel2 encapsulates logic from Illustration 29. Guard [30] (a boolean expression, that should evaluate to true for the transition to fire) on the Generator transition ensures the desired amount of cased (2000 in this case) shall be executed. An inscription on the arc from Generator to starting place increments the trace number by one and increments the time by a randomized value with a mean value of one hour using the custom *IAThour()* function. Transition inscription on the Generator transition passes the trace number *i* further and produces the variable(s) to decide, which submodel shall handle the case (this is done by drift function, explained at the next step). The purpose of next two unnamed transitions is to route a process instance to the correct model using transition guards. Another example is in Illustration 31 where there are four substitution transitions and thus four possible process variants.

In this model the guard of the Generator transition also discards the traces, started not during the work hours. This model was used for generating logs with multi-order dynamics.

3. Configure concept drift settings of the orchestrating model using drift functions (a function with model time or trace number and concept drift parameters as input and the ordinal number of the submodel to handle the case as output):
    1. For sudden drift use the *chooseSudden(t1, i)* function, where *t1* is desired drift point and *i* is the trace id. Example: to acquire a log with a sudden drift occurring at trace number 1200, we should use *chooseSudden(1200, i)*.
    2. For gradual drift with a linear distribution the randomized *chooseLinear(t1, t2, i)* where *t1* is the start of the drift period, *t2* is the end of that period and *i* is the trace id. Example: to acquire a log with a linear gradual drift occurring from trace number 400 to trace number 600, we should use *chooseLinear(400, 600, i)*.
    3. For gradual drift with an exponential distribution use the randomized *chooseExponential(t1, t2, i, halflife)* function, where *t1* is the start of the drift period, *t2* is the end of that period, *i* is the trace id, *halflife* is the period after which the probability of occurring of previous process is half of the original. Note that in the description of exponential graduality in the preliminaries section, it was noted that exponential gradual drift does not end. In practice we end the drift explicitly. There will be a cutoff at trace *t2* when probability reaches *0.5^((t2-t1)/halflife)*. Example: to acquire a log with an exponential gradual drift occurring from trace number 500 with cutoff at trace number 1000 and where probability should be halved every 300 traces we should use *chooseExponential(500, 1000, i, 300)*.
    4. For sudden drift with multi-order dynamics use the non-randomized functions *chooseYearHalf(time())*, *chooseWeekend(time())*, *choose6WeekPeriod(time())*, *choose24WeekPeriod(time())* etc. Other similar functions could be easily defined. Output of the function fully depends on the model time, e.g. if the time belongs to the first half of the year, *chooseYearHalf* shall return 1, otherwise it will return 2. The integer return by built-in *time()* function is considered as the

number of seconds passed from 01.01.1970 00:00 (also known as UNIX time). Example: to acquire a log as on Illustration 6, so that micro-level drifts are every 6 weeks and macro-level drifts are ever 24 weeks, use *choose24WeekPeriod(time())* as the first drift function and *choose6WeekPeriod(time())* as the second drift function.

4. Run the simulation until no transitions are enabled.
5. Import the simulation report into ProM (as CPN Tools Simulation Log).
6. Run "Convert Key/Value Set to Log" plug-in with following mapping (as in Illustration 32):
   - Trace Identifier - cpnvariable:i (where *i* is our trace id variable in the model)
   - concept:name (Concept) - Transition
   - time:timestamp (Time) – Time
7. Run "Filter log using Simple Heuristics" plug-in. In the "Event filter" dialog choose only actual events. Exclude auxiliary events (joins, splits, generator etc). Refer to Illustration 33
8. Analyze the log using "Concept Drift" plug-in.

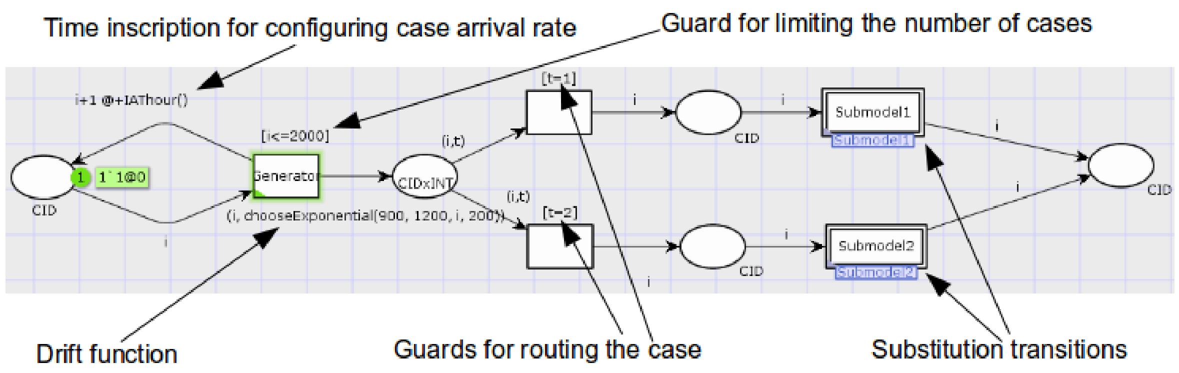


Illustration 30: Orchestrating model for gradual concept drifts

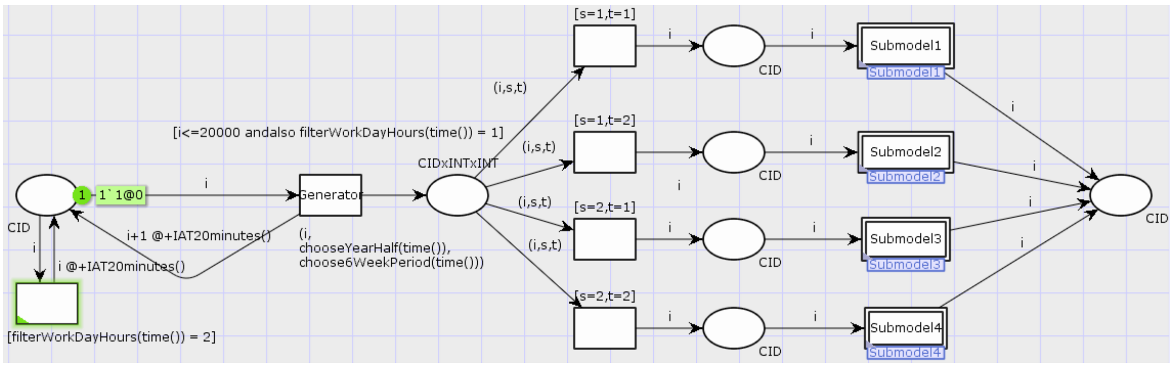
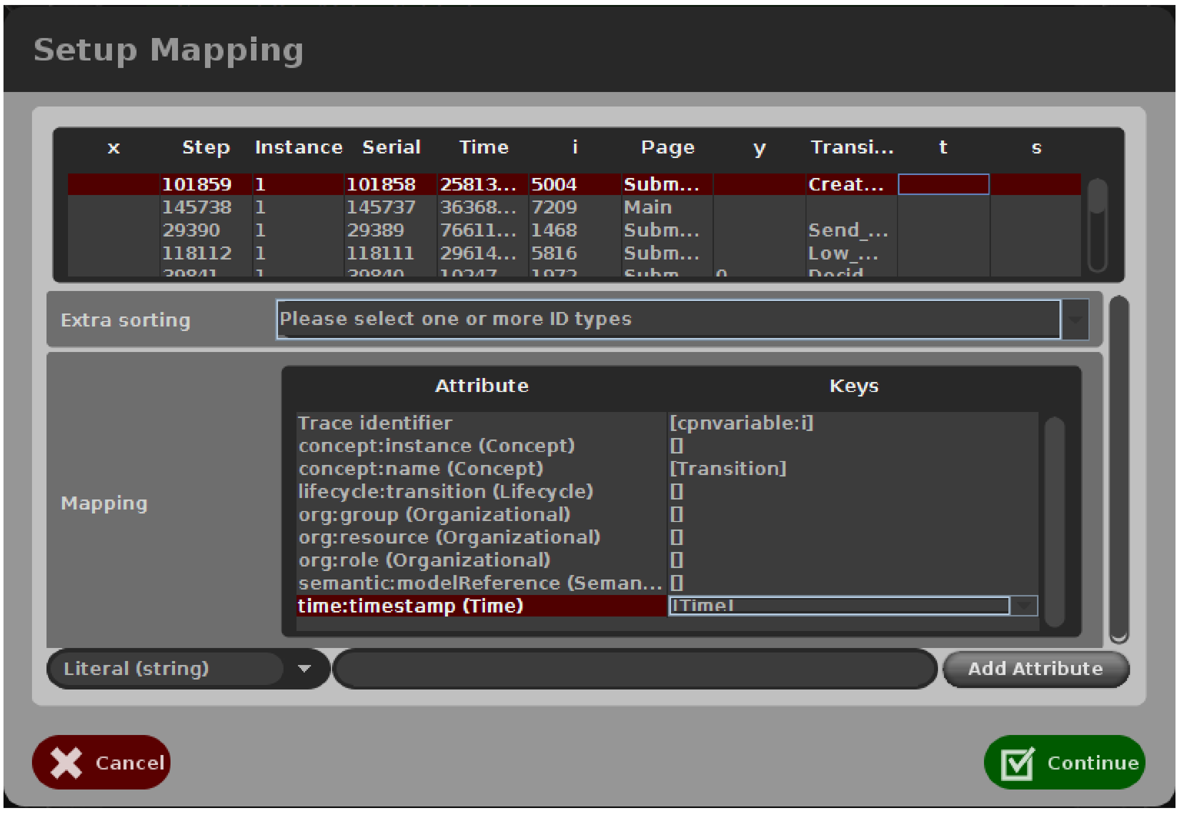


Illustration 31: Orchestrating model for multi-order dynamics


Setup Mapping

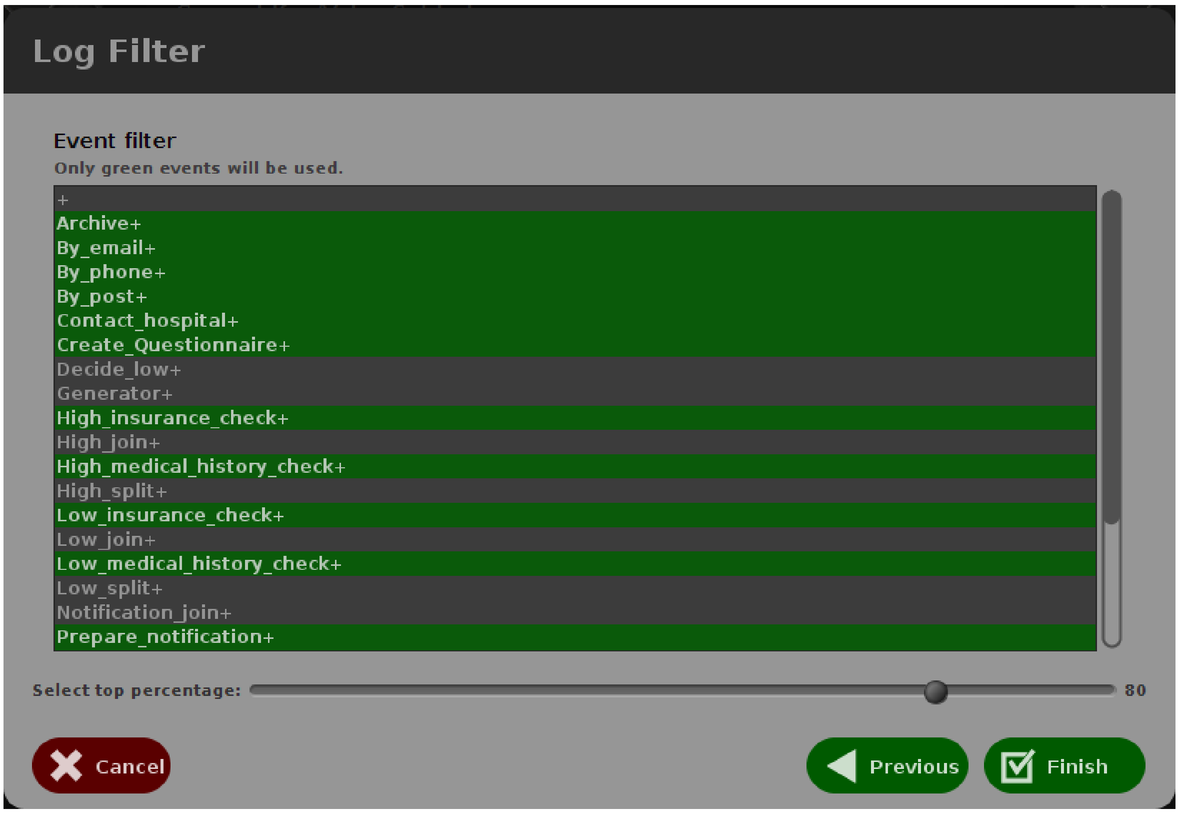
x	Step	Instance	Serial	Time	i	Page	y	Transi...	t	s
	101859	1	101858	25813...	5004	Subm...		Creat...		
	145738	1	145737	36368...	7209	Main				
	29390	1	29389	76611...	1468	Subm...		Send_...		
	118112	1	118111	29614...	5816	Subm...		Low_...		
	29841	1	29840	10247	1972	Subm	0	Decid		

Extra sorting: Please select one or more ID types

Mapping

Attribute	Keys
Trace identifier	[cpnvariable:i]
concept:instance (Concept)	[]
concept:name (Concept)	[Transition]
lifecycle:transition (Lifecycle)	[]
org:group (Organizational)	[]
org:resource (Organizational)	[]
org:role (Organizational)	[]
semantic:modelReference (Seman...	[]
time:timestamp (Time)	[Time]

Illustration 32: Configuring the mapping in the "Convert Key/Value Set to Log"


Log Filter

Event filter

Only green events will be used.

```
+
Archive+
By_email+
By_phone+
By_post+
Contact_hospital+
Create_Questionnaire+
Decide_low+
Generator+
High_insurance_check+
High_join+
High_medical_history_check+
High_split+
Low_insurance_check+
Low_join+
Low_medical_history_check+
Low_split+
Notification_join+
Prepare_notification+
```

Select top percentage: 80

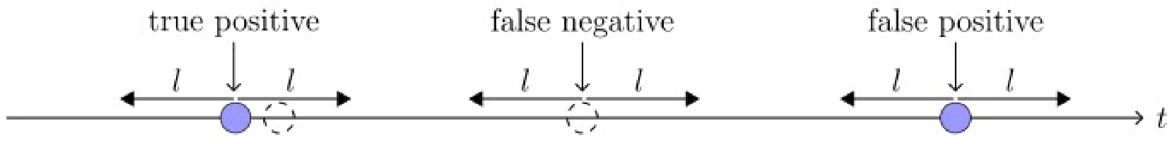
Illustration 33: Filtering the log by keeping the significant event and excluding the auxiliary ones

Quality Evaluation Framework

To assess the quality of the algorithms, we use the common metrics used in the field of information retrieval: true positives (TP), false negatives (FN), false positives (FP) and some derived metrics. Note that true negatives (TN) are not defined as it is difficult to define the correct absence of the result in our case. First we need to choose a lag size *l*. A *lag* is the size of the error allowed for the algorithm, given in trace numbers. Example: there is a concept drift in the log at trace 300, but the algorithm detected it at point 330. If the lag size is 50, then we consider this detection as a hit (correct discovery), but if the lag size is 20, then we do not consider that as a hit. Using the lag size *l*, the metrics are defined as follows (see also Illustration 28):

- TP: a drift is detected at point *i* and there is an actual drift in $i \pm l$
- FN: there is an actual drift at point *i* and there is no drift detected in $i \pm l$
- FP: a drift is detected at point *i* and there is no actual drift in $i \pm l$
- Precision: describes the probability of the detected drift to be actual $\frac{TP}{TP+FP}$

- Recall: describes the probability of an actual drift to be detected $\frac{TP}{TP+FN}$
- F1-score: a harmonic mean of precision and recall $\frac{2 * precision * recall}{precision + recall}$



Ideally, the lag size *l* should be chosen so that there can not be two actual drifts separated by less than *l* traces. If more than one drift is detected within *l* traces from the actual drift, then the closest hit becomes TP, and all of the other become FP. Example: assume there is an actual concept drift at 600 and the lag size is 50. If the algorithm detects three drifts at points 580, 610 and 630, then 610 would be considered as TP, while 580 and 630 would be considered as two FPs.

Computational Complexity Evaluation

To assess the computational complexity and its dependency on input parameters we took the insurance claim log from [8] and ran both the original algorithm with different step sizes and the ADWIN with variable population minimum and maximum sizes, while all other input parameters stayed fixed. All runs used J-measure over all activity pairs as a feature extracted from traces and Kolmogorov-Smirnov test as the hypothesis test. Every test configuration was run 5 times to provide average computational time together with 95% confidence intervals.
Illustration 28 shows how the run time of the original algorithm decreases when the step size is increased. One can clearly see that run time and step size are inversely proportional. Illustration 29 shows how adjusting the minimum and the maximum population sizes in ADWIN affects the run time complexity. The varying of population minimum size has almost no effect on the run time, while increasing the maximum population size increases the run time, however they are not directly proportional.

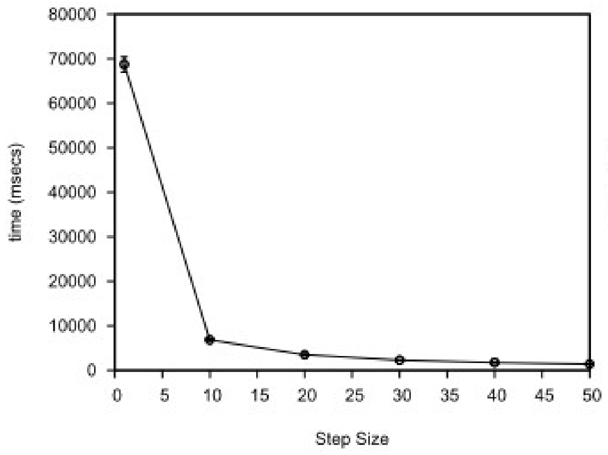


Illustration 35: Influence of step size on run time of original algorithm

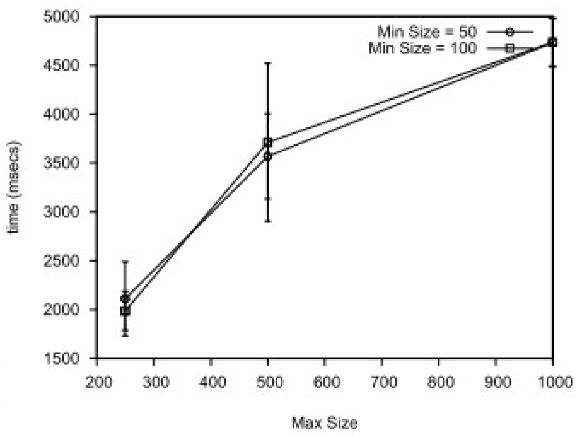


Illustration 36: Influence of min and max population sizes on run time of ADWIN

Increasing the step size increases the amount of valuable information we lose (from the hypothesis tests we do not perform), thus we can not make it too big. To evaluate this, we applied the original algorithm (where step size is 1) and the same algorithm with step sizes k=20 and k=200 to the insurance claim log from [8] using J-measure, Kolmogorov-Smirnov test and population size of 400 traces. Compare the p-value plots in Illustration 30 and Illustration 31. Increasing the step size to 20 traces keeps all of the information about the concept drifts. Increasing the step size to 200 outputs the result shown in Illustration 32. One can still see 4 concept drift troughs, however, a lot of information is lost.

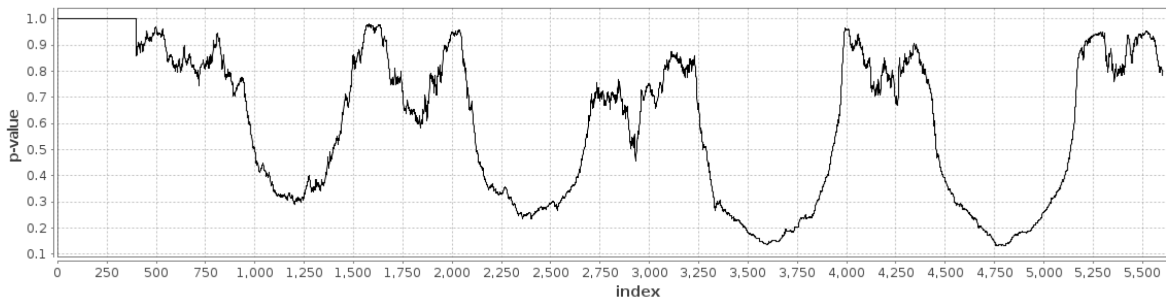


Illustration 37: Step size = 1 trace

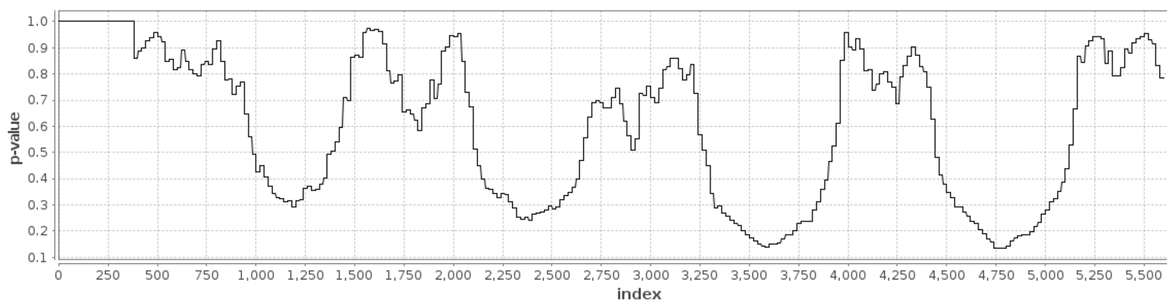


Illustration 38: Step size = 20 traces

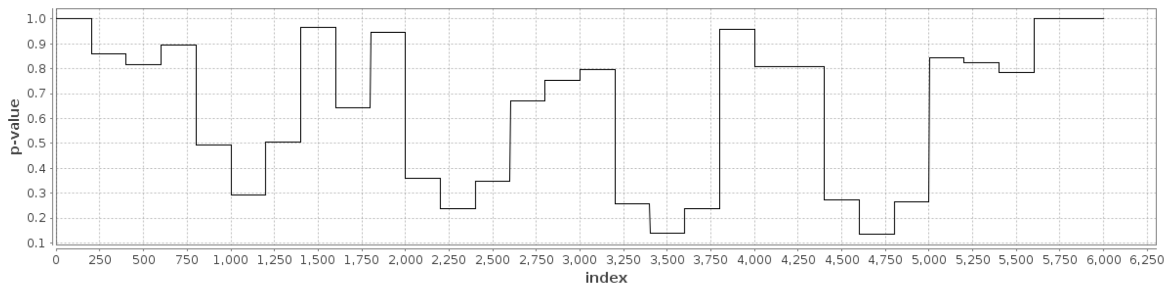


Illustration 39: Step size = 200 traces

Sudden Drifts

To assess the quality of change point detection algorithm, we ran ADWIN on the original log with following input parameters: feature = J-measure for every activity pair, hypothesis test = Kolmogorov-Smirnov test, p-value threshold = 0.4, minimum population size = 100, maximum population size = 500, step size = 20.

The result of the experiment is depicted on Illustration 40. The change points are detected at indices 1207, 2415, 3598, and 4793. Using a lag size of 20 traces gives us a perfect result: TP=4, FP=0, FN=0, recall=precision=F1-score=1.0.

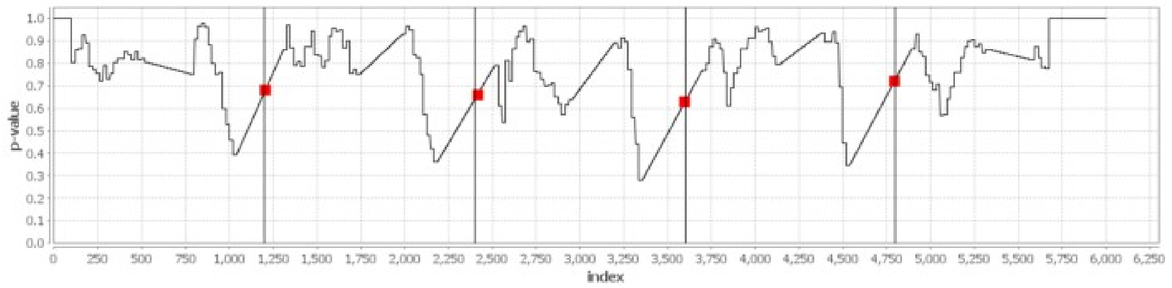
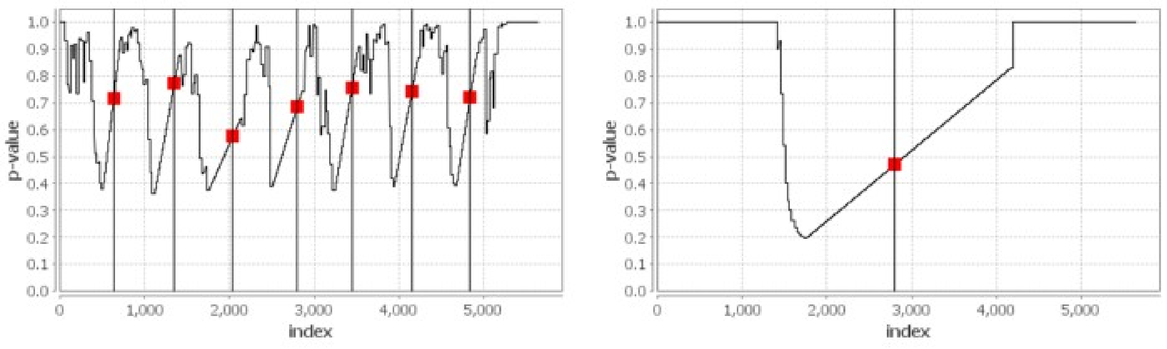


Illustration 40: Detection of sudden drifts. Red dots indicate automatically detected drifts, vertical lines indicate actual drifts.

Multi-Order Dynamics

To assess the quality of detecting multi-order dynamics, we generated a log, which exhibits changes as in Illustration 6, thus there are 4 different process variants of the remodeled insurance claim process with minor drifts occurring every 6 weeks and major drifts occurring every 24 weeks. For simplicity, a fixed arrival rate of approximately 3 cases per hours was chosen. Cases started only during working hours on week days. The log has 5647 cases and 57530 events. There are 7 micro-level drifts at points 629, 1346, 2038, 2802, 3444, 4156, 4845 and one macro-level drift at 2802.

For detecting micro-level drifts a minimum population size of 3 days, maximum population size of 6 weeks, step size of 1 day, and a p-value threshold of 0.4 was chosen. For detecting macro-level drifts a minimum population size of 12 weeks, maximum population size of 24 weeks, step size of 3 days, and a p-value threshold of 0.2 was chosen. The resulting p-value of the KS-test on the J-measure over all activity pairs plots of the experiments are depicted on Illustration 41 and the table actual vs detected micro-level concept drift is on Illustration 42. It shows that all the micro-level drifts were detected with perfect accuracy. Macro-level drift was also detected at the correct index (2802).



(a) micro-level drifts (b) macro-level drift

Illustration 41: Detection of multi-order changes. Red dots indicate detected drifts, vertical lines indicate actual drifts

	change point						
	1	2	3	4	5	6	7
actual drift index	629	1346	2038	2802	3444	4156	4845
detected drift index	629	1346	2038	2802	3444	4156	4845

Illustration 42: Actual vs detected micro-drift change points

P-value Threshold Selection Analysis

As you may have noticed, every proposed algorithm depends on the chosen p-value threshold heavily. To analyze how the quality of change point detection depends on the chosen p-value threshold, we conducted the following experiment: we took the log with multi-order dynamics, and ran the ADWIN to detect micro-level drifts. The experiment was run 12 times, with p-value thresholds chosen from 0.15 to 0.7. For evaluation, a lag size of 2 days (50 traces) was chosen. The result of the experiment is in Illustration 43. When we increase the p-value threshold (thus lowering the bar for a drift to be detected), then the amount of false positives grows, thus the precision decreases. At the same time the amount of false negatives declines, thus the recall increases. The F1-Score, which incorporates both precision and recall grows until it reaches maximum, when threshold is 0.4, and decays after that. We have figured out, that for this log, the best p-value threshold is 0.4. It is not guaranteed, that this will hold for every other log, but 0.4 is a good value to start the experiments, thus we selected it as a default value in the ProM Concept Drift plug-in.

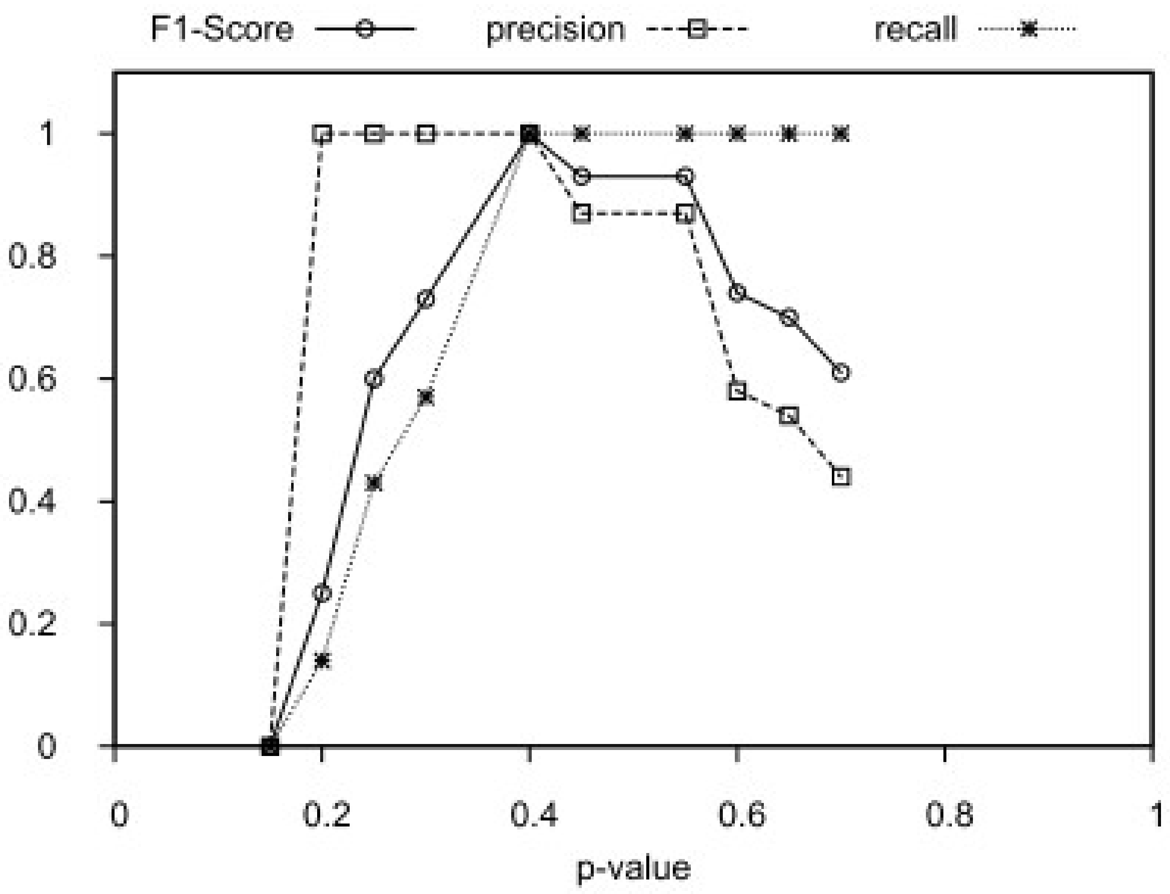


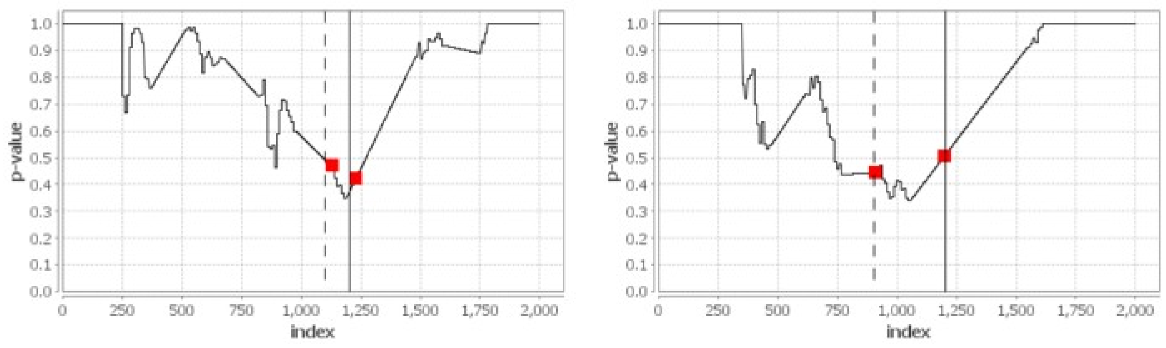
Illustration 43: Influence of p-value threshold on change detection metrics

Gradual Drifts

For evaluation of gradual drift detection we generated two logs from the remodeled insurance claim log. One log consisted of 2000 traces and 19346 events and incorporated a linear gradual drift between traces 1100 and 1200. Second log consisted of 2000 traces and 19183 events and incorporated an exponential gradual drift starting from trace 900 with probability half-life of 100 and cut-off at trace 1200.

For the log with a linear gradual drift the result of using gradual ADWIN for the KS-test on the J-measure over all activity pairs using a minimum population size of 200, maximum population size of 300, step size of 10, gap size of 100, and a p-value threshold of 0.35 is depicted on Illustration 44(a). A drift was detected with a change period from 1128 to 1229. Repeating the experiment with a gap of size 50 (while all other arguments staying the same) outputs a drift with a change period from 1158 to 1199.

For the log with an exponential gradual drift the result of using gradual ADWIN for the KS-test on the J-measure over all activity pairs using a minimum population size of 200, maximum population size of 300, step size of 10, gap size of 300, and a p-value threshold of 0.35 is depicted on Illustration 44(b). A drift was detected with a change period from 907 to 1198. Repeating the experiment with a gap of size 100 (while all other arguments staying the same) outputs a drift with a change period from 907 to 998.



(a) linear gradual drift (b) exponential gradual drift

Illustration 44: Detection of gradual drifts. The red dots indicate the detected start/end points of the change period, the interval between dashed and solid vertical lines indicate the actual change period

Analyzing Real-life Data

Description of the Real-life Log/Process

To test the algorithm on real-life data, we use the log from Business Processing Intelligence Challenge (BPIC) [31]. According to the description on the website, it is "a real-life log, taken from a Dutch Financial Institute. This log contains some 262.200 events in 13.087 cases. Apart from some anonymization, the log contains all data as it came from the financial institute. The process represented in the event log is an application process for a personal loan or overdraft within a global financing organization. The amount requested by the customer is indicated in the case attribute AMOUNT_REQ, which is global, i.e., every case contains this attribute. The event log is a merger of three intertwined sub processes. The first letter of each task name identifies from which sub process (source) it originated from." First trace in the log starts on 01.10.2011 and the last trace starts on 01.03.2012, thus the log is exactly half a year long.

There were 6 submissions to the jury, but nobody has analyzed the log for concept drifts. We do not provide an objective quality evaluation framework, because we do not know for sure, whether there were any concept drift at all.

Results

We shall look for sudden drifts using ADWIN with time periods and the KS-test on the J-measure over all activity pairs. First we shall look for micro-level drifts by choosing the following input parameters: p-value threshold = 0.4, minimum period = 1 day, maximum period = 1 month, step period = 12 hours. The result is in Illustration 45. The algorithm has detected a drift at trace 10580. Drift date is Feb 5, 2012 11:24:30 PM.

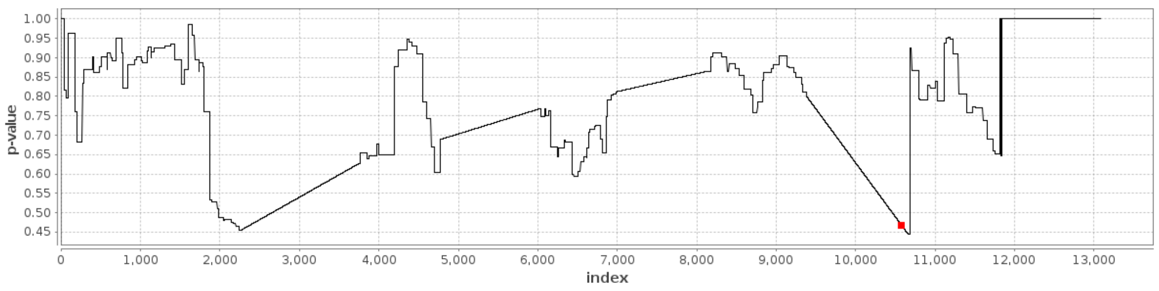


Illustration 45: Detecting micro-level drifts in BPIC log

Now we shall look for macro-level drifts by choosing the following input parameters: p-value threshold = 0.4, minimum period = 1 month, maximum period = 3 months, step period = 3 days. With these parameters, the algorithm hasn't detected a single, so we increased the p-value threshold to 0.45 while leaving other parameters the same. The result is in Illustration 46. The algorithm has detected a drift at trace 4757. Drift date is Nov 23, 2011 5:42:30 PM.

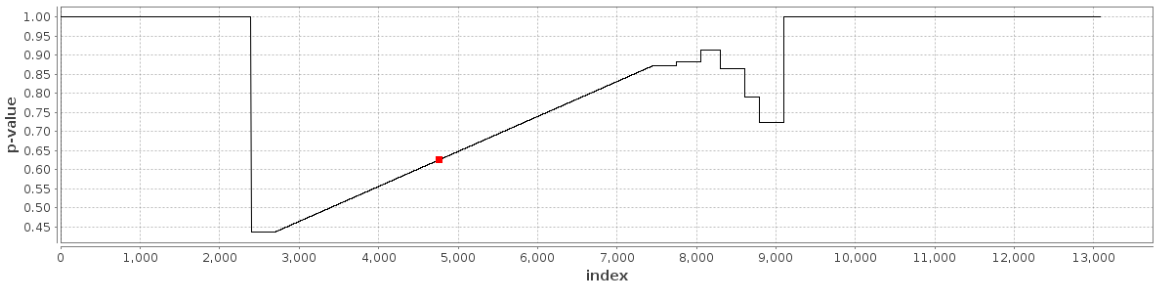


Illustration 46: Detecting macro-level drifts in BPIC log

Unfortunately we were not able to localize the change or unravel the process evolution, which leads to the need of efficient approaches for dealing with those two problems.

6. Conclusions and Future Work

In this thesis we proposed five novel approaches to deal with concept drift in process mining, which were tested on both simulated and real-life data. Step size improvement showed wonderful results in speeding up the algorithm without any downsides in quality. Automatic change point detection extracted all of the change points with a reasonable error margin. ADWIN algorithm has reduced the dependency on the population size, thus lowering the amount of false negatives and false positives. Defining the size of populations of consecutive traces using time periods showed perfect results in detecting concept drift in logs with multi-order dynamics and the same idea could probably be exploited in other process mining algorithms, that deal with populations (windows) of traces. Finally, ADWIN with non-continuous populations and gap shrinking algorithm showed its possibility to detect gradual drifts, though choosing the appropriate gap size input parameter can be crucial for getting adequate results.

The method of generating logs with concept drift and the quality evaluation framework have the value on their own, because they can be used by anyone, who is conducting research in detecting concept drift in process mining and who needs to generate test data or needs a way to evaluate results.

There is a lot of work yet to be done in the field of detecting concept drift in business processes. Both the algorithms proposed in this thesis and the ones, described in Related Work section are only capable of detecting changes from the control-flow perspective, thus there is a need for algorithms that also consider data and resource perspectives. The problems of change localization and unraveling the process evolution also need more attention. An exhaustive research can be done in how existing drift detection algorithms cope with different process change patterns described in [17].

Hypothesis test based approach from [8], expanded and improved in different ways in this thesis, still has many possible evolution paths. One can consider some completely different features that could be extracted from the traces or a way for comparing the populations without executing hypothesis tests.

Summary

Process mining is a relatively new research area, but it is already used in practice. Every company and organization run different business processes, which are supported by information systems and which leave event logs while being executed. By analyzing those logs one can build a process model, which reflects how the process operates in reality. Surprisingly, this can differ dramatically from the existing comprehension. Applying different process mining algorithms can help in discovering process optimization possibilities, detecting fraudulent execution and comparing similar processes in different organizations. Existing algorithms assume that the analyzed process is in steady state, however it could be altered because of seasonality, a new law or some event, like a financial crisis. In this case, we have to deal with concept drift. Concept drifts can be sudden, when the change is abrupt and gradual, where one concept fades gradually while the other takes over.

The most significant work in this area was done by Bose et al. [8]. It involves extracting features, e.g. J-measure, from process execution traces and executing statistical hypothesis tests between fixed-size populations of consecutive trace feature values. The result of such algorithm is a plot of p-values, which needs to be manually inspected for any concept drift.

In this work we proposed five novel approaches for detecting concept drifts in process mining. All of them improve or expand the algorithm, proposed in [8]. Step size improvement allows to speed up the algorithm from [8] by leaving out some intermediate steps. Automatic change point detection algorithm allows to extract the concept drift points without the need to analyze the plot manually. The adaptive windows algorithm (ADWIN) relaxes the original algorithm's dependency on the fixed population size, thus reducing the amount of false positives and false negatives. The algorithm with non-continuous populations allows to deal with gradual drifts. And finally, defining the population sizes in terms of time periods instead of trace amount allows to detect micro-level and macro-level drifts in logs with multi-order dynamics, where process changes can happen on multiple level of granularity. The algorithms were implemented in the Concept Drift plug-in of ProM framework.

For assessing the quality of algorithms, we proposed a way to generate logs with different

concept drift characteristics using CPN Tools and a quality evaluation framework, similar to the one used in the field information retrieval, involving calculating true positives, false positives, false negative and derived metrics.

Test results on simulated data were promising. From the quantitative side, the step size improvement could speed up the algorithm 20 times without losing in quality. Also the dependency of run time on the ADWIN input parameters was examined. From the qualitative side, the algorithm has perfectly detected all the sudden drifts in both logs with and without multi-order dynamics. Also an experiment for discovering the optimum p-value threshold (an input parameter for ADWIN) was conducted. In the logs with gradual drifts, change periods were detected, but the quality was not perfect.

Resümee

Efektiivsed algoritmid kontseptsiooninihke leidmiseks äriprotsessides.

Magistritöö (30 EAP)

Jevgeni Martjušev

Protsessikaeve on suhteliselt uus, kuid ühiskonna poolt juba kasutusele võetud uurimisvaldkond. Paljud ettevõtted ja asutused rakendavad erinevaid infosüsteemidega toetatud protsesse, mille käivitamisest jäävad maha sündmuste logid. Neid logisid analüüsides saab ehitada mudeli, mis kajastab, kuidas need protsessid reaalselt toimivad. Üllataval kombel võib tegelik protsessi täitmise olukord tugevalt erineda oodatust. Erinevate protsessikaeve algortimide rakendamine lubab leida protsessides optimeerimisvõimalusi, välja selgitada väärkasutuse juhtumeid, võrrelda sarnaseid protsesse erinevates organisatsioonides. Tänapäevased algoritmid eeldavad, et analüüsitav protsess on stabiilne, kuid tegelikult võib seda mõjutada hooaegsus, uus seadus või mõni väline sündmus – näiteks järsk majanduslangus. Sellisel juhul on tegemist kontseptsiooninihkega. Kontseptsiooninihked võivad olla järsud (kui protsessi muutus on äkiline) või järkjärgulised (kui üks protsessivariant asendub teisega sujuvalt).

Kõige olulisem senine uurimistöö antud valdkonnas on tehtud Bose ja teiste poolt [8]. See hõlmab tunnuste (nt *J-measure*) ekstraheerimist protsessi käivitamisel salvestatud andmetest ning statistiliste hüpoteeside kontrollimist järjestikuste populatsioonide vahel. Sellise algoritmi tulemuseks on p-väärtuste graafik, kust tuleb käsitsi leida kontseptsiooninihe.

Antud töös pakkusime välja viis uudset lähenemist kontseptsiooninihke avastamiseks protsessikaeves. Igaüks neist parandab või laiendab algset Bose poolt kirjeldatud algoritmi [8]. Sammu pikkuse suurendamine võimaldab algoritmi kiirendada, jättes välja mõned vahepealsed sammud. Muutmispunkti automaatne leidmine võimaldab ekstraheerida kontseptsiooninihke punktid ilma manuaalse analüüsita. Adapteerivate akende algoritm (ADWIN) pehmendab originaalse algoritmi sõltuvust populatsiooni suurusest, seega vähendab vale-positiivsete ja vale-negatiivsete tulemuste arvu. Mittejärjestikkuste populatsioonidega algoritm võimaldab uurida järkjärgulisi kontseptsiooninihkeid. Lisaks

lubab populatsioonide suuruste määramine ajaliste perioodide kaupa (jälgede koguse asemel) leida mikro-taseme ja makro-taseme nihked multi-taseme dünaamikaga logides, kus protsess muutub mitmel detailsuse tasemel. Kõik algoritmid olid implementeetirud ProM raamistiku Concept Drift moodulis.

Algoritmide kvaliteedi hindamiseks pakub käesolev töö välja meetodi, kus CPN Tools programmi abil genereeritakse logisid erinevate kontseptsiooninihke tunnustega. Samuti on välja arendatud kvaliteedi hindamise raamistik, mis sarnaneb sellega, mis on kasutusel infootsingu valdkonnas ning mis hõlmab endas tegelike positiivsete, valepositiivsete ja valenegatiivsete väärtuste loendamist ning tuletatud meetrikate arvutamist.

Testide tulemused simuleeritud andmetel on paljulubavad. Kvantitatiivse poole pealt võimaldas sammu pikendamine kiirendada algoritmi tööd 20 korda ilma kaotuseta kvaliteedis. Samuti uuriti tööaja sõltuvust ADWIN sisendparameetritest. Kvalitatiivse poole pealt avastas algoritm kõik kontseptsiooninihked, sh ka multi-taseme dünaamikaga logides. Samuti viidi läbi eksperiment optimaalse p-väärtuse künnise avastamiseks. Järkjärguliste nihketega logides suudeti küll avastada muutmispiirkonnad, kuid kvaliteet ei olnud täiuslik.

References

[1] Sarbanes, P., G. Oxley et. al.: Sarbanes-Oxley Act of 2002 (2002)

[2] van der Aalst, W.M.P., Rosemann, M., Dumas, M.: Deadline-based Escalation in Process-Aware Information Systems. Decision Support Systems 43(2), 492–511(2011)

[3] van der Aalst, W.M.P., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T. et al.: Process mining manifesto. In Business process management workshops (pp. 169-194). Springer Berlin Heidelberg (2012)

[4] ProM website: http://www.promtools.org/prom6/

[5] CPN Tools website: http://cpntools.org/

[6] van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag New York Inc (2011)

[7] Bradford, L., Dumas, M.: Getting started with YAWL (2007)

[8] Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I., Pechenizkiy, M.: Handling Concept Drift in Process Mining. In: CAiSE. LNCS, vol. 6741, pp. 391–405.Springer, Berlin (2011)

[9] Definition of Petri Net on http://www.informatik.uni-hamburg.de/TGI/PetriNets/faq/

[10] van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of Circuits Systems and Computers, 8:21–66 (1998)

[11] Jensen, K., Kristensen, L.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer (2009)

[12] Standard ML website: http://www.standardml.org/

[13] Pechenizkiy, M., Bakker, J., Žliobaitė, I., Ivannikov, A., Kärkkäinen, T.: Online Mass Flow Prediction in CFB Boilers with Explicit Detection of Sudden Concept Drift. SIGKDD Explorations 11(2), 109–116 (2009)

[14] Schlimmer, J., Granger, R.: Beyond Incremental Processing: Tracking Concept Drift. In: Proceedings of the Fifth National Conference on Artificial Intelligence. vol. 1, pp. 502–507 (1986)

[15] Widmer, G., Kubat, M.: Learning in the Presence of Concept Drift and Hidden Contexts. Machine Learning 23(1), 69–101 (1996)

[16] Bifet, A., Gavaldà, R.: Learning from Time-Changing Data with Adaptive Windowing. In: SDM. pp. 443–448 (2007)

[17] Weber, B., Rinderle, S., Reichert, M.: Change Patterns and Change Support Features in Process-Aware Information Systems. In: CAiSE. LNCS, vol. 4495, pp.574–588.

Springer (2007)
[18] Ploesser, K., Recker, J.C., Rosemann, M.: Towards a Classification and Lifecycle of Business Process Change. In: BPMDS. vol. 8 (2008)
[19] Regev, G., Soffer, P., Schmidt, R.: Taxonomy of Flexibility in Business Processes. In: BPMDS. Citeseer (2006)
[20] Günther, C.W., Rinderle-Ma, S., Reichert, M., van der Aalst, W.M.P.: Using Process Mining to Learn from Process Changes in Evolutionary Systems. International Journal of Business Process Integration and Management 3(1), 61–78 (2008)
[21] Definition of p-value on iSixSigma. http://www.isixsigma.com/dictionary/p-value/
[22] Carmona, J., Gavaldà, R.: Online Techniques for Dealing with Concept Drift in Process Mining. In: IDA. LNCS, vol. 7619, pp. 90–102 (2012)
[23] Luengo, D., Sepúlveda, M.: Applying Clustering in Process Mining to Find Different Versions of a Business Process That Changes over Time. In: BPM Workshops(1). LNBIP, vol. 99, pp. 153–158 (2012)
[24] Bose, R.P.J.C.: Process Mining in the Large: Preprocessing, Discovery, and Diagnostics. Ph.D. thesis, Eindhoven University of Technology (2012)
[25] Weber, P., Bordbar, B., Tino, P.: Real-Time Detection of Process Change using Process Mining. In: Imperial College Computing Student Workshop. Department of Computing Technical Report, vol. DTR11-9, pp. 108–114 (2011)
[26] Martjushev, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Change Point Detection and Dealing with Gradual and Multi-Order Dynamics in Process Mining Background.
[27] De Medeiros, A.A., Günther, C.W.: Process mining: Using CPN tools to create test logs for mining algorithms. In Proceedings of the sixth workshop on the practical use of coloured Petri nets and CPN tools (CPN 2005), Vol. 576 (2005)
[28] ProM KeyValue plug-in manual:
http://westergaard.eu/wp-content/uploads/2011/07/KeyValue.pdf
[29] CPN Tools manual, Hierarchy:
http://cpntools.org/documentation/concepts/hierarchy/start
[30] CPN Tools manual, Guards:
http://cpntools.org/documentation/concepts/colors/inscriptions/guards
[31] BPIC website: http://www.win.tue.nl/bpi2012/doku.php?id=challenge

Appendices

A. ProM Concept Drift Plug-in

To run ProM with the Concept Drift plug-in you need to download ProM from http://www.promtools.org/prom6/, run the ProM's package manager, install ConceptDrift package and then run ProM itself. Refer to the "Getting started" guide in case of problems. The source code for ProM Concept Drift plug-in is accessible from SVN: https://svn.win.tue.nl/repos/prom/Packages/ConceptDrift/Trunk/.

B. Models

CPN Tools models used to generate the logs with gradual drifts and logs with multi-order dynamics are on the CD.

C. Logs

The insurance claim process logs are on the CD. The loan application process log is available from the website of 8th International Workshop on Business Process Intelligence 2012: http://www.win.tue.nl/bpi2012/doku.php?id=challenge.