Detecting Concept Drift in Processes using Graph Metrics on Process Graphs

Alexander Seeliger
Technische Universität Darmstadt,
Telecooperation Lab
Darmstadt, Germany
seeliger@tk.tu-darmstadt.de

Timo Nolle
Technische Universität Darmstadt,
Telecooperation Lab
Darmstadt, Germany
nolle@tk.tu-darmstadt.de

Max Mühlhäuser
Technische Universität Darmstadt,
Telecooperation Lab
Darmstadt, Germany
max@tk.tu-darmstadt.de

ABSTRACT

Work in organisations is often structured into business processes, implemented using process-aware information systems (PAISs). These systems aim to enforce employees to perform work in a certain way, executing tasks in a specified order. However, the execution strategy may change over time, leading to expected and unexpected changes in the overall process. Especially the unexpected changes may manifest without notice, which can have a big impact on the performance, costs, and compliance. Thus it is important to detect these hidden changes early in order to prevent monetary consequences. Traditional process mining techniques are unable to identify these execution changes because they usually generalise without considering time as an extra dimension, and assume stable processes. Most algorithms only produce a single process model, reflecting the behaviour of the complete analysis scope. Small changes cannot be identified as they only occur in a small part of the event log. This paper proposes a method to detect process drifts by performing statistical tests on graph metrics calculated from discovered process models. Using process models allows to additionally gather details about the structure of the drift to answer the question which changes were made to the process.

CCS CONCEPTS

•Information systems →Data mining; •Applied computing →*Business intelligence; Enterprise data management;*

KEYWORDS

Change Point Detection; Concept Drift; Process Drift; Process Mining; Process Dynamics

ACM Reference format:
Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. 2017. Detecting Concept Drift in Processes using Graph Metrics on Process Graphs. In *Proceedings of 9th International Conference on Subject-oriented Business Process Management, Darmstadt, Germany, March 30 - 31, 2017 (S-BPM ONE '17),* 10 pages.
DOI: 10.1145/3040565.3040566

S-BPM ONE '17, Darmstadt, Germany

978-1-4503-4862-1/17/03…$15.00
DOI: 10.1145/3040565.3040566

1 INTRODUCTION

The analysis of event logs of process-aware information systems (*PAISs*) using *Process Mining* [21] has become very popular. Process Mining aims to extract valuable knowledge from recorded event logs and consists of three major tasks: discovery, conformance checking and enhancement of process models [21]. Many different algorithms have been proposed to discover "as-is" process models from event logs, delivering a precise picture of the current execution strategy of processes. These process models allow the analysis of the actual use of process-aware systems, thus providing an internal and objective perspective (e.g. "Where is a bottleneck in the process?" and "Where does the process differ from the ideal world?") of the process. However, a major issue of most process mining approaches is the assumption of a stable process over time [9], which does not hold in most cases.

In comparison to the data mining and machine learning communities where *concept drift* [26] is a well known problem, *process drift* is a relatively rare research topic in process mining. Current process discovery algorithms do not consider *permanent* or *temporary* changing processes which results in inaccurate process models. However, processes are constantly influenced by many different hidden contexts such as seasonal changes, new regulations or organisational changes. We cannot assume that processes behave stable over a longer period of time because those different influences have an impact on the process execution behaviour. Currently, such process drifts cannot easily be detected or extracted with state of the art process mining algorithms. For example, discovery algorithms are applied over an event log that usually consists of data over a longer period of time, thus these algorithms generalise over the considered time span and do not provide a precise representation of the actual process execution strategy. So resulting process model will either additionally contain such changes, leading to more complex process models, or they are completely hidden because the occurred change is too little. In order to retain this knowledge it is important to localise process drifts in event logs to improve the quality of automatically discovered process models.

Process drifts can be characterised as *planned* and *documented* (e.g. changes due to a change of guidelines or regularities), and *unexpected* (e.g. changes due to new employees who do not perform work as desired or changes due to a change of resource capacity which may affect the process by reducing the amount of normally required tasks) behaviour changes. In case of a planned process drift, organisations can check whether the desired change is correctly executed or if, for example, further employee training is required. Detecting unexpected behaviour changes helps organisations to quickly identify process risks and helps them to take actions on

critical executions to prevent monetary consequences (e.g. violations against governmental regulations). Process drifts can occur on all four process mining perspectives: functional, control-flow, organisational and data process [5]. In this paper we will focus on the concept drift in the control-flow perspective.

This paper proposes an algorithm that uses statistical significance tests to localise process drifts in event logs. We use different graph metrics (e.g. number of nodes/edges, node centrality, network degree etc.) calculated from discovered process models as the input to detect process drifts. Process models are calculated for smaller sub-logs of the event log thus we can compare the graph metrics for different time spans. By using an adaptive window approach we can automatically determine an optimal size of the smaller sub-logs to detect process drifts in event logs without prior knowledge of the event log. Additionally, our approach extracts further knowledge about detected process drifts such as which events in the event log are modified, removed or added which is neglected by other related work. Using our approach retains behavioural changes to the process, allowing to detect expected and unexpected behaviour in event logs over a longer period of time and can automatically determine what changes have been made to the process.

The paper is structured as follows. In section 2 we introduce related work. Section 3 presents the approach for detecting process drifts in event logs. In section 4 we evaluate our approach using generated synthetic event logs. Section 5 discusses and concludes our work.

2 RELATED WORK

Although much work is related to make processes more flexible [17, 18] or to help organisations to optimise their processes, most work in process mining assumes a stable process [9]. While concept drift is well known in the data mining and machine learning community, little work related to process mining can be found.

In [6] the authors use process model change logs to mine changes of processes in PAISs. The presented approach allows to analyse the influences of process changes by providing an aggregated view of all happened changes. However, having such a change log implies that the approach is unable to identify hidden process drifts. Changes must be documented in the change log in order to be able to analyse them.

Bose et al. [2, 3] only use event logs as the input to identify process drifts by introducing different global and local features. However, the approach is not fully automated and it is unable to detect all classes of changes. The user needs some knowledge about the event logs and the process drifts in order to be able to correctly set the required window size in which the algorithm searches for process drifts. Setting the optimal size of the window heavily influences the accuracy of the approach. Maaradji et al. [12] use an adaptive sliding window approach to overcome the issue of selecting the optimal window size. They perform statistical hypothesis testing on partial ordered runs to determine whether a process drift occurred by comparing two consecutive windows. A run is a higher level representation of traces which considers the concurrency of events in the event log. A modified feature set was used in [16] to detect drifts from event streams in unpredictable business processes. Instead of using complete runs Ostovar et al. use the α^+ relations to model the behaviour of the process which defines five relations: conflict, concurrency, causality, length-two loop and length-one loop. Manoj Kumar et al. [13] use event correlation strength instead of runs to perform process drift detection.

A different approach is used in [1, 8]. The authors aim to detect process drifts by clustering event log traces. For each trace the distances between each event pairs are calculated, thus the structure of the process regarding the order of events is considered for drift detection. Here, also a window size has to be set which influences the number of detected process drifts. Carmona et al. [4] propose a real-time approach which learns internal representations of the event log. The approach estimates the faithfulness of this representation using an adaptive window approach to automatically detect process drifts in event logs.

In [10] Lakshmanan et al. determine differences between two sets of traces by performing a graph spectral analysis. The graph model is generated from the distances between trace vectors which represent the connectivity between events. However, the authors do not consider analysing the change of processes over time.

Related work shows that there exist some approaches addressing the detection of process drifts. In comparison with related work we use discovered process models to detect drifts which allows to provide explicit explanations for each drift. None the related work addresses this problem yet.

3 PROCESS DRIFT DETECTION

Detecting process drifts in an time-sorted event log requires us to characterise what a drift actually is. We define a process drift as a significant behavioural change of the process execution that occurred over some time (it may be temporary or permanent) and that most traces in an event log follow. In comparison to the normal distribution of different traces (e.g. noisy event logs) we are looking for a process drift that influences almost all traces (e.g. an event is not executed any more). *Trace equivalence* which is a popular notion of equivalence in the process mining community is too sensitive to detect process drifts. It compares all possible traces in a process model which would lead to a process drift if any of the possible traces have slightly changed. Besides not being too sensitive to smaller changes we also need to take care of concurrent events. For example two traces may be equivalent if two events are executed in parallel although the traces are different regarding the order of events. If there exists a trace where a is followed by b and another trace where b is followed by a, then both traces are not the same although the process is the same because a and b might be executed in parallel. This is why we use process models discovered by a process mining algorithm to determine a significant behavioural change of the process because the algorithm takes care of concurrency and noisy event logs.

Our idea is to compare the structure of "as-is" process models to identify process drifts. These precise process models can be easily extracted from event logs by using existing process mining discovery algorithms which overcome the issues to determine equivalence of processes. For example, if we begin skipping an event at some point in time and create a process model before and after this change, we will see that the change is also reflected in one of the discovered process models. To detect process drifts we look

for graph-metrics changes over time. Another advantage of using process models is that we use the model to extract the structure of the change without much more calculation because we already gather the process model.

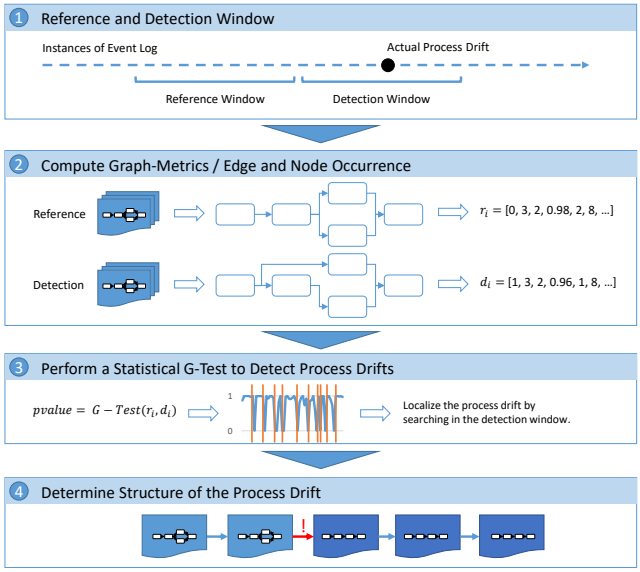

Figure 1: Overview of our Process Drift Detection Approach

Our approach works on top of event logs and is divided into four main steps (cf. figure 1) which are repeated until the whole event log is scanned for process drifts:

(1) First we use two consecutive adaptive windows to split the given event log into two smaller sub-logs (reference and detection window),
(2) Second we calculate process models from these two resulting sub-logs and calculate various graph-metrics (e.g. edge and node occurrence, graph-degree, number of edges, node-degree etc.) from the process model,
(3) Then we perform a statistical G-test between the graph metrics of the two consecutive sub-logs to detect process drifts and
(4) Lastly, we extract the differences of the process models to characterise what has actually changed.

In the following sections we will describe the four steps of our approach in detail.

3.1 Splitting Event Log into Reference and Detection Window

An event log consists of multiple traces which again consists of multiple events. Each trace represents a single instance of a process execution and specifies in which order events have been executed. Let us consider that we have an time-sorted event log $W \subseteq T^*$ where T is a set of events. An event log consists of a set of traces $\rho \in W$, namely ordered lists of events $\rho = t_0, t_1, t_i, \ldots, t_n$ with $i \in \{0, 1, \ldots, n\}$. n is varying for each trace as each trace can have different lengths.

In order to detect process drifts in the event log, we perform statistical significance tests over graph metrics of adjacent process models which are generated using a process mining discovery algorithm. The statistical significance test uses two populations of the same size from two consecutive sub-logs of the event log to determine significant differences in the distribution. In the following, we will call the first sub-log the *reference window* and the second sub-log the *detection window* [12]. Reference and detection window are adjacent to each other but they are not overlapping (cf. figure 1 (1)). They build a composite window of $2 \cdot w$ traces of the event log such that $P_1 = \rho_0, \rho_1, ..., \rho_w$ and $P_2 = \rho_{w+1}, \rho_{w+2}, ..., \rho_{2w}$ where ρ_i are traces in W. Using the statistical significance test we evaluate the hypothesis whether the population of the reference window (P_1) is similar to the population of the detection window (P_2). If this is not the case we assume a process drift between the reference and the detection windows. To detect all process drifts in the event log, we iteratively move both windows over the traces in the event log until all traces have been visited. Each time the significant test evaluates negative (both populations are not similar) we mark the point between the reference and the detection window as a potential process drift.

Because we have no prior knowledge about the event log, finding the right windows size w is an essential part of our approach because it highly influences the accuracy. So we use an adaptive window that automatically adjusts itself (lines 29-39). We start our approach by using a window size of 100 traces and increase the window size by the factor 1.2 if the result of the statistical test is negative. If we reached the maximum window size (we use 200 traces), we consider that there is no process drift present in the event log for the scanned traces and move the reference window to the start of the detection window which also reduces the computation time. We shrink the window size again to the minimum size and repeat the statistical test. If the statistical test is positive, we consider a process drift in the detection window.

```
1  while (i < |W| - wSize) {
2    pval = test(W[i:i + wSize],W[i+wSize:i+wSize*2])
3
4    if (pval < threshold) {          // a potential drift was found?
5      found = false
6
7      newWSize = wSize / 2
8      lastI = i + wSize * 2
9
10     for (j = i + wSize - newWSize;
11       j < i + wSize * 2 - newWSize) {
12
13       lastI = i + newWSize * 2
14
15       pval2 = test(W[j:j+newWSize],          // perform g-test on
16         W[j+newWSize:j+newWSize*2])          // smaller windows
17
18       if (pval2before - pval2 < -0.5)
19           break
20
21       pval2before = pval2
22
23       if (pval2 < threshold) {
24         changePoints <- j + newWSize * 2   // change point found
25         i = j + newWSize * 2
26         found = true
27       }
28     }
29     if (!found) {
30       wSize = wSize * lastI / (i + wSize * 2)
31       i = i + wSize
```

```
32      }
33    } else {
34      wSize = wSize * 1.2       // increase window size
35    }
36    if (wSize >= maxSize) {
37      wSize = 100               // no change point found
38      i = i + maxSize           // reset and set new reference window
39    }
40  }
```

Listing 1: Pseudocode of the adaptive window approach.

To specify the position of the process drift more precisely we repeat the statistical testing with reduced window sizes (lines 10-28) and restrict the search scope to the detection window. The current window sizes are divided by two and kept fix. Both smaller windows are moved over the restricted search scope to refine the position of the potential detected process drift. If the statistical test is positive we mark this point as a process drift, otherwise we repeat testing in the search scope. If the statistical test is never positive we adjust the window size of the original windows and move both windows one window size forward. To reduce the amount of statistical testing we use an early stopping heuristic (line 18) which depends on the p-value of the statistical tests.

3.2 Compute Graph-Model and Metrics

After the event log is split into a reference and a detection window, we can discover process models for both windows by applying an existing process mining discovery algorithm. For this work we use the heuristics miner [24] because it delivers a good abstraction of the event log, it is fast and it automatically handles the concurrency of events. The heuristics miner takes an event log as the input and produces a heuristic net which reflects the dependencies between events by analysing the *followed by* relation. Let $a, b \in T$ two different events from an trace $\rho \in T^*$. If a is directly followed by b then $|a >_w b|$ is the number of times where a is directly followed by b. The dependency graph can be built by using the following formula:

$$a \Rightarrow_w b = \frac{|a >_w b| - |b >_w a|}{|a >_w b| + |b >_w a| + 1}$$

A high value of $A \Rightarrow_w B$ is an indicator for a strong dependency relation between A and B. As this value is calculated for each possible event combination, we can construct a dependency graph that reflects the execution strategy of events. Nodes correspond to events and edges correspond to the dependency relation *"directly followed by"*. The heuristics miner additionally contains heuristics to detect loops, AND/XOR-splits/joins, non-observable tasks and long distance dependencies, which will not be explained here but can be found in [24]. The result of the heuristics miner is a graph model that represents the "as-is" execution strategy based on the observed traces and the dependencies between events.

We compute the process model using the heuristics miner for the reference and the detection window. The basic idea is that if we can observe a change of the process model within two consecutive windows the process execution strategy has changed. Minor changes and noisy event logs are automatically handled by the heuristics miner so the resulting process model will contain no or only minor changes thus our approach is also robust against those issues. We determine the deviation of both observed process models by applying a statistical significance test over different graph metrics:

- Number of nodes / edges
- Graph Density
  $D = \frac{|E|}{|V| \cdot (|V|-1)}$
- In- and out-degree of each node
- Occurrence of node / edges

Each graph metric is an indicator for a specific change in the process execution strategy. The number of nodes indicates if the number of executing events has changed, either new events were added or events were skipped. A change of the number of edges or of the graph density is an indicator for a more or less complex process model due to an in- or decrease of process variants over time. The in- and out-degree of each node allows us to identify the events which have mostly changed, and which control flows have been added or removed. Lastly, the occurrence of nodes and edges can be used to determine if the distribution over each event and transition has changed. With these graph metrics we are able to describe any change of the process model to identify process drifts in the event log.

$$M^* = \begin{array}{c} \\ W^*_{reference} \\ W^*_{detection} \end{array} \begin{array}{c} \begin{array}{cccc} m_0 & m_1 & \dots & m_n \end{array} \\ \begin{pmatrix} 1 & 5 & \dots & 0.99 \\ 3 & 5 & \dots & 0.95 \end{pmatrix} \end{array}$$

All computed metrics are summarised in two vectors (for reference and detection window) which contain the values for each metric $m_i \in M$. The size of the vector is specified by the number of nodes in the graph.

3.3 Perform a G-Test on the Graph-Metrics

After we have gathered the graph-metrics for the reference and the detection window, we perform a statistical G-Test to determine if the process model of the detection window is significantly different to the model of the reference window. The G-test [14] is a distribution free maximum likelihood statistical significance test which can be used to compare observed distributions with expected distributions. We use the occurrence of edges as the input for the statistical tests which is the number of traces that follow this specific edge. The statistical null hypothesis is that the observed distribution over the edges in the detection window is equal to the distribution in the reference window. The formula for calculating the p-value of the statistical test is the following:

$$G = 2 \sum_i O_i \cdot \ln \frac{O_i}{E_i}$$

where O_i is the observed occurrence of an edge and E_i is the expected occurrence under the null hypothesis. The sum is only calculated for all non-zero counts. The result of the G-test is the significance probability (P-value). A possible process drift is detected if the probability is less than the significance level α (threshold).

In our experiments it turned out that performing statistical tests over the occurrence of edges works better than any other graph-metric mentioned above. However, we keep all graph-metrics as they are used later to determine the structure of the change (see section 3.4). For the α value we used 0.0001 as the significance level

which provided the most accurate results. It turned out that using the occurrence of edges is a very precise indicator for process drifts thus we can select a really low significance level leading to a low change of a false positive.

For each possible detected process drift we perform additional statistical tests using different reference and detection windows (see section 3.1) to localise the position of the process drift more precisely. In addition to the significance test of the distribution of the edges, we also test the significance of the distribution of observed nodes. If both significance tests are positive then our approach marks the position between the reference and the detection window as a process drift. After detected a process drift, we move the reference and detection window forward to detect the next drift in the event log until all traces have been visited.

3.4 Determine the Process Drifts

After we have identified the locations of the process drifts we gather more details about the structural change and what modifications were detected within the transition from the reference to the detection window. We gather this knowledge by comparing the graph metrics from the previous step. Due to the fact that the process model and their graph metrics are already calculated, we do not need to perform much additional work to extract the structural changes.

In the first step we gather the difference of the in- and out-degree of nodes in the graph to determine which events have changed between the reference and the detection window. This allows us to determine how the graph has changed, for example, which edges or nodes have been added or removed from the graph. For any change that happened to the in- and out-degree of nodes we collect the number of traces that follow the changed edge of this node to determine how many traces now follow a different path through the graph. Using this information we can determine the effect of the detected process drift and return this information to the user.

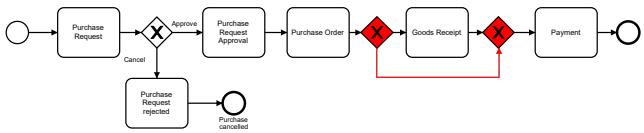

Figure 3: Example process model with a modification (marked as red).

Let us consider a simple example process (see figure 3) where a new edge from *Purchase Order* to *Payment* was added to the process graph while the event *Goods Receipt* is now skipped. The example is a simple procurement process where an *Purchase Request* is *Approved* before an *Purchase Order* can be created. After the order was made, a *Goods Receipt* is created before the *Payment* is executed. In the example the event *Goods Receipt* is now bypassed thus after the order is created the invoice is directly paid without creating a goods receipt. When applying our described approach we will get the following graph metric changes:

```
NETWORK DEGREE  = -1.0
NUMBER OF EDGES = -2.0
NUMBER OF NODES = -1.0
INDEGREE:  Goods Receipt  = -1.0
INDEGREE:  Payment        = -1.0
OUTDEGREE: Purchase Order = -1.0
OUTDEGREE: Goods Receipt  = -1.0

Purchase Order -> Goods Receipt = -16.0 (0.0)
Goods Receipt  -> Payment       = -16.0 (0.0)
Purchase Order -> Payment       =  14.0 (24.0)
```

In the given example we can see that there are two edges less and one node less after the detected process drift. The in-degree of the nodes *Goods Receipt* and *Payment* has decreased by 1 which means that two incoming edges were removed. The out-degree of the nodes *Purchase Order* and *Goods Receipt* has also decreased by 1 which means that two outgoing edges were removed. Now one would wonder why the in- and out-degree of *Purchase Order* and *Payment* have changed although only a node was removed in the before and after process model. In the given example our approach has detected the process drift at trace number 1020 (original drift was at 1000), thus the before model additionally contains 20 traces that already have changed. The result is that the before model is build like depicted in figure 1 including the red marked modifications and in the after model the node *Goods Receipt* is missing. However, we can still extract the exact change that happened.

We can use change of the amount of traces that follow the edges in the graph: On the one hand the drift has reduced the amount of traces on the edge *Purchase Order* to *Goods Receipt* from 16 to 0 and on the edge *Goods Receipt* to *Payment* from 16 to 0. On the other hand we can see an increase of traces on the edge *Purchase Order* to *Payment* from 10 to 24. From all this collected information we can reason that there were two edges from *Purchase Order* to *Goods Receipt* and from *Goods Receipt* to *Payment* removed Now more traces follow the edge *Purchase Order* to *Payment*.

This additional knowledge can help analysts to evaluate the process drift and explains why a specific point in time was marked as a process drift. Instead of just returning the process drift our approach delivers an explanation which also allows further analysis.

4 EVALUATION

We implemented our proposed approach as a ProM [23] plug-in and used the implementation to evaluate the performance of our approach. ProM is a framework developed by the University of Eindhoven for researchers to rapid prototyping process mining specific algorithms and methods. Our plug-in reads the given event log and automatically determines process drifts and extracts the change (e.g. the modifications to the process model) that happened before and after the drift. The output of the plug-in is a list of potential process drifts, the corresponding process models and the structural change that happened.

4.1 Setup

To determine the performance of our approach, we measure the *F1-score* which is the harmonic mean of recall and precision. The F1-score is an indicator whether our approach can correctly identify the process drifts in the given event log (precision returns the probability that a detection is correct and recall is the probability

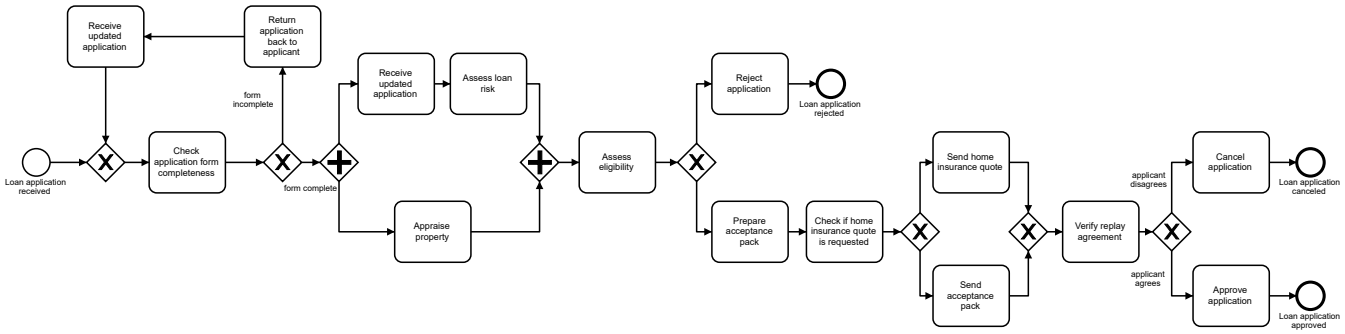

Figure 2: Example process model of the base model of the synthetic event log [12] used in the evaluation.

that we recognised a correct change [7]). Additionally, we calculate the *average delay* between the actual and the detected process drift which is an indicator for how early our approach is able to detect an actual change.

For evaluating our approach we used synthetic event logs (see figure 2) to determine the accuracy of our approach. We used the same benchmark of 72 event logs used in [12] where different parameters were varied. Event logs were generated from a base model which consists of 15 events and was modified systematically to generate process drifts. These modifications are classified into different change patterns (cf. table 1) and categorised into insertion ("I"), resequentialisation ("R") and optionalisation ("O"). To create more complex change patterns, the simple change patterns were combined thus resulting in additional event logs ("IOR", "IRO", "OIR", "ORI", "RIO", "ROI").

Table 1: Change patterns of the synthetic event logs [12].

#	**Change Pattern**	
re	Add/remove fragment	I
cf	Make two fragments conditional/sequential	R
lp	Make fragment loopable/non-loopable	O
pl	Make two fragments parallel/sequential	R
cb	Make fragment skippable/non-skippable	O
cm	Move fragment into/out of conditional branch	I
cd	Synchronise two fragments	R
cp	Duplicate fragment	I
pm	Move fragment into/out of parallel branch	I
rp	Substitute fragment	I
sw	Swap two fragments	I
fr	Change branch frequency	O

Each synthetic event log is composed of a fixed number of alternating instances generated from the base model, followed by a fixed number of instances of the modified model, such that each used event log consists of exact 9 process drifts. Also the size of the event logs was varied: 2500, 5000, 7500 and 10000. The event logs are annotated with the drifts such we can easily calculate precision, recall and average delay.

4.2 Accuracy Results

We imported the set of synthetic event logs into ProM and used our plug-in to compute the process drifts. Figure 4 reports the F1-score for each change pattern averaged over the four different event log sizes compared with existing approaches [2, 3, 12]. For Bose et al. a fixed window size of 100 was used and for Maaradji et al. the superior adaptive window approach was used.

We can see that for 16 out of 18 change patterns our approach gathers an F1-score between 1.0 and 0.9 (cf. table 2). We gather an overall F1-score of 0.9466. For 11 change patterns our approach delivers better performance than the related work. Only for the change patterns *lp* and *OIR* related work is significantly better than our approach. For all other change patterns our approach is almost as good as the approaches compared. Further analysis reveals that for change pattern *lp* we only gather F1-scores above 0.87 for log size 2500 and 5000. For the larger size event logs our approach only delivers F1-scores of 0.61. It turns out that the significance test does not deliver a high probability thus the detection does not consider these points as process drifts.

In figure 5 the mean distance between the actual and the detected process drift is depicted. As our approach performs a second statistical hypothesis test on the potential process drifts, we can optimise the position of the process drift. We can also see that the difference is very small and for some cases we perform better than the related work (*cb, cm, cp, fr, IOR, IRO, RIO, ROI, rp*).

In summary, our approach delivers a high performance regarding the F1-score (above 0.9) and the average distance (23.65) in the used synthetic event logs. In comparison with the related work we gather a better performance for 11 event logs and a lower average distance for 9 event logs.

4.3 Reason Extraction Results

Besides the actual detection of process drifts in event logs our approach provides additional information about the drift such as which events and/or edges have been added or removed. This information can help to understand the actual process drift allowing to perform further analysis. To evaluate the reasons for process drifts extracted using our approach, we manually compared the changes made to the event logs (based on the process model that was used to add process drifts to the event logs) with the results of our approach. Because each change pattern (cf. table 1) has made

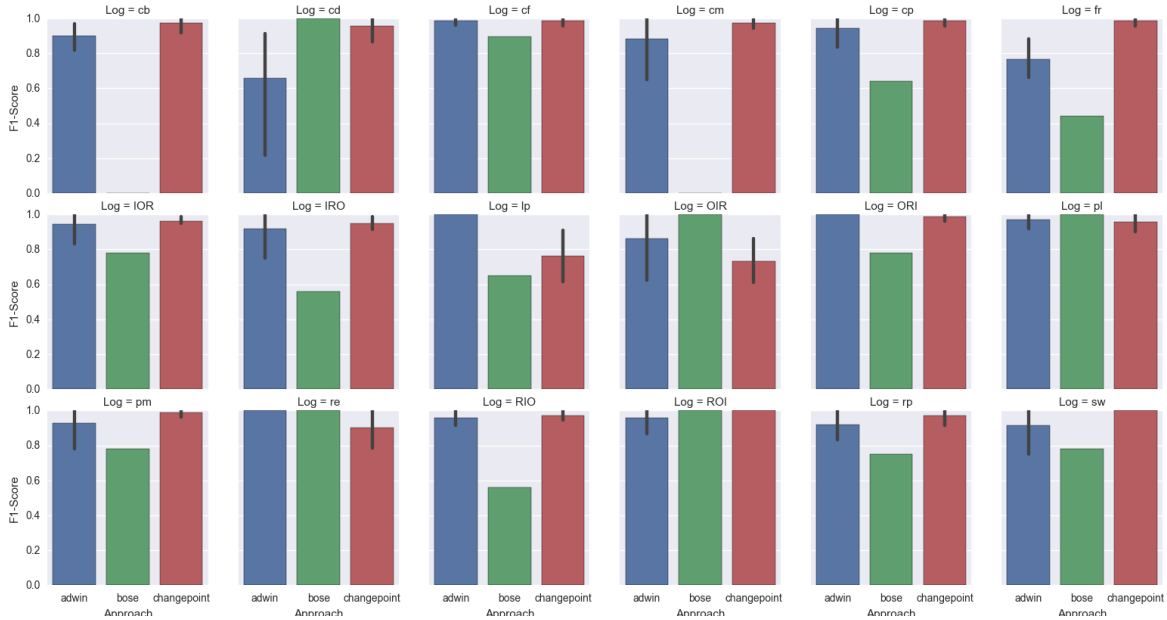

Figure 4: F1-score for different change patterns (higher is better) and compared with Bose et al. (green) [3] and Maaradji et al. (blue) [12]. Lines in the graph correspond to the value range which have been observed for different event log sizes.

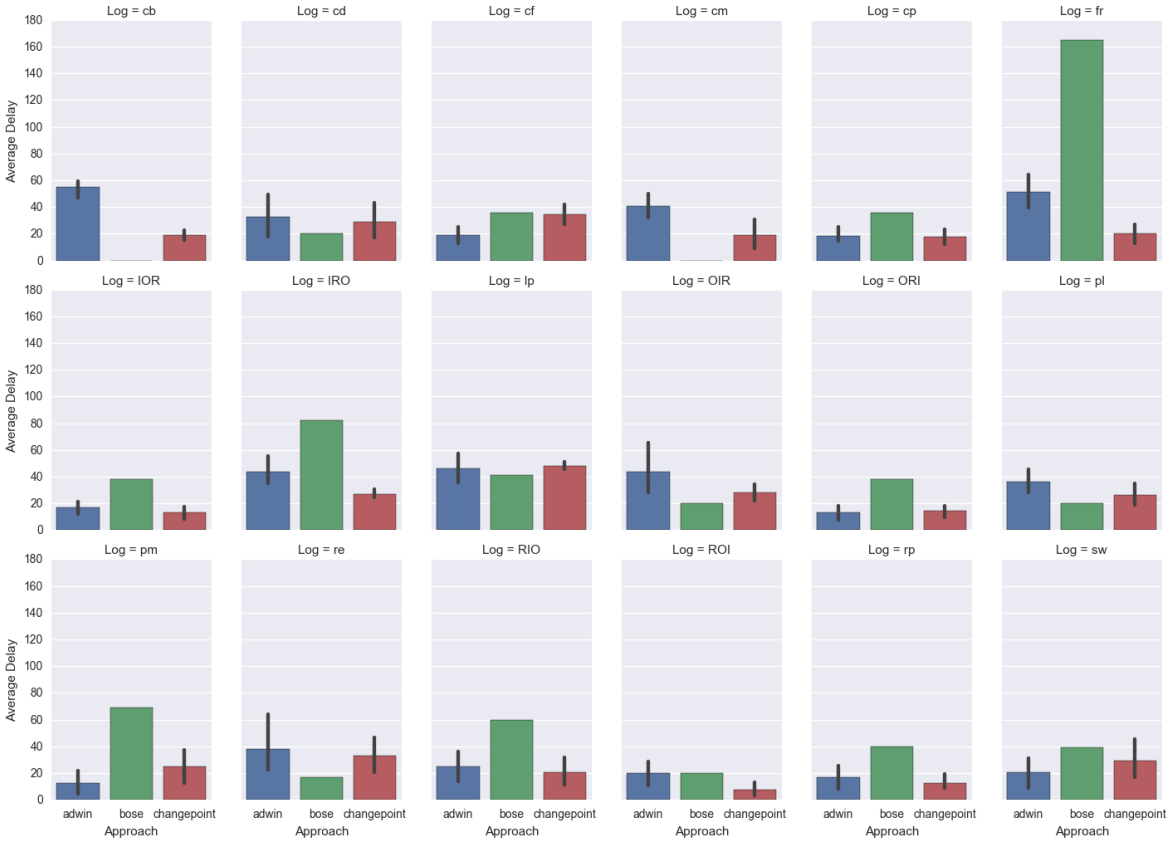

Figure 5: Average delay for different change patterns (lower is better). Lines in the graph correspond to the value range which have been observed for different event log sizes.

Table 2: Synthetic dataset: Results of the process drift detection methods compared to related work. It shows the average F1-score and the mean distance.

#	Change Point		Bose [3]		Adwin [12]	
	F1	Dist.	F1	Dist.	F1	Dist.
fr	**0.9853**	**19.92**	0.4420	165.00	0.7677	51.56
cb	**0.9722**	**18.94**	-	-	0.8985	54.72
cd	0.9546	28.69	**1.0000**	**20.00**	0.6599	32.35
cf	0.9853	34.62	0.8950	36.00	**0.9868**	**19.08**
cm	**0.9722**	**19.24**	-	-	0.8833	40.91
cp	**0.9853**	**17.59**	0.6395	36.00	0.9444	18.65
IOR	**0.9606**	**13.00**	0.7805	38.00	0.9444	16.71
IRO	**0.9487**	**27.22**	0.5612	82.00	0.9167	43.81
lp	0.7618	48.03	0.6484	**41.00**	**1.0000**	46.25
OIR	0.7331	28.06	**1.0000**	**20.00**	0.8603	43.67
ORI	0.9869	14.25	0.7805	38.00	**1.0000**	**12.97**
pl	0.9575	26.33	**1.0000**	**20.00**	0.9722	36.53
pm	**0.9869**	24.78	0.7805	69.00	0.9265	**12.87**
re	0.9036	33.02	**1.0000**	**17.00**	**1.0000**	38.11
RIO	**0.9722**	**20.77**	0.5612	60.00	0.9591	25.08
ROI	**1.0000**	**7.31**	**1.0000**	20.00	0.9559	19.79
rp	**0.9722**	**12.67**	0.7500	40.00	0.9194	16.99
sw	**1.0000**	29.61	0.7805	39.00	0.9167	**20.79**
avg.	**0.9466**	**23.56**	0.7011	41.17	0.9173	30.60

different changes to the process we can identify what structural changes can be correctly identified by our approach.

For convenience we use a detailed evaluation example to show the results of our approach on a single change pattern and then show the results of all change patterns summarised. Let's take the *cb* change pattern which added a new conditional edge to the process model. In figure 6 we can see that an additional edge was created from the event *Assess eligibility* to the end of *Check if home insurance quote is requested*. Our approach delivers the following reasons for the observed process drift:

```
OUTDEGREE Assess eligibility=1.0,
OUTDEGREE Check if home insurance qu... =  -1.0

Assess eligib...->Prepare acceptance... =  -8.0 (16.0)
Assess eligib...->Send home insuranc... =  10.0 (10.0)
Assess eligib...->Reject application    =  -7.0 (26.0)
Check if home...->Send home insuranc... = -12.0  (0.0)
Check if home...->Send acceptance pa... =  -3.0  (9.0)
```

From the computed result we can see that the events *Assess eligibility* and *Check if home insurance quote is requested* were involved in the process drift. The edge from *Assess eligibility* to *Send home insurance quote* is completely new (increase from 0 to 10) and the edge from *Check if home insurance quote is requested* to *Send home insurance quote* is completely removed. In this case a new conditional branch was added to the process model with allows to bypass events *Prepare acceptance pack* and *Check if home insurance quote is requested*.

Table 3 shows the reasons for each change pattern extracted using our approach. We mark a reason correct (✓) if the modification to the event log correspond to the events identified as the reasons for the process drift. If our approach was unable to find the correct reason then we marked this as an error (✗).

Table 3: Reasons extracted from our approach for each change pattern.

#	Reason	
re	Remove of *Assess eligibility*	✓
cf	*Send acceptance pack* and *Send home insurance quote* are sequential	✓
lp	-	✗
pl	*Check credit history, Assess Loan risk* and *Appraise property* are sequential	✓
cb	New edge from *Check if home insurance quote is requested* to *Send home insurance quote*	✓
cm	*Check if home insurance quote is requested* followed by *Prepare acceptance pack*	✓
cd	New edge *Appraise property* to *Assess loan risk* Remove edge *Appraise property* to *Assess eligibility*	✓
cp	Detection of a loop	✗
pm	New edge *Check if home insurance quote is requested* to *Prepare acceptance pack* New edge *Prepare acceptance pack* to *Verify repayment agreement*	✓
rp	*Replace events*	✓
sw	Remove edge *Send acceptance pack* to *Verify repayment agreement* Remove edge *Assess eligibility* to *Prepare acceptance pack* Remove edge *Send home insurance quote* to *Verify repayment agreement* Add edge *Send home insurance quote* to *Prepare acceptance pack* Add edge *Send acceptance pack* to *Prepare acceptance pack* Add edge *Assess eligibility* to *Verify repayment agreement*	✓
fr	-	✗

In three cases our approach is unable to extract the correct reason for the process drift. In two cases our approach could not find any reason at all. For all other simple change patterns we can provide additional information that helps to understand the process drift.

5 DISCUSSION AND FUTURE WORK

In this paper, we proposed an automatic process drift detection approach which uses process models discovered from event logs to identify and localise drifts. Our main contribution is the usage of graph metrics to localise process drifts in event logs and that our approach provides further information about the structural change that was detected. We described the four steps of our approach in detail and how all these components work together to deliver accurate process drift detection results. In our evaluation we showed

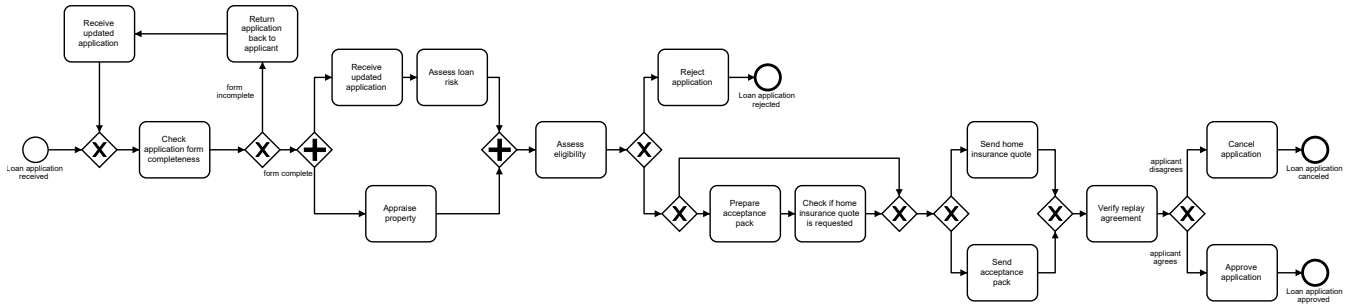

Figure 6: Process model of the *cb* change pattern event log [12].

that we can achieve really good results using our approach and that the provided information about detected process drifts is able to describe the actual change.

However, there is also some room for improvement for the presented approach. One possible improvement is to change to a different process mining discovery algorithm to a more recent and robust method. It turns out that the heuristics miner is not able to correctly identify long loops thus we cannot detect process drifts that are caused because of loops in the process execution. By using a different discovery approach, we can improve the detection of process drifts without having to change any other components [11, 25]. Additionally, we need to investigate in process drifts that do not occur suddenly but slowly. We think that the approach could potentially detect such changes when increasing the maximum window size. However, this will have an influence on the detection of sudden drifts. One possible future research direction here is trying to maximize the size of the reference window which could potentially improve the detection of slowly increasing drifts. However, finding the right larger window size is quite challenging as we have no prior knowledge about the event log, thus we do not know if the process has not been altered for a longer time.

One drawback of the current implementation of our approach is the lack of visualisation for the extracted process drift reasons. A textual representation of the changes is not very convenient and a more visual approach would help to better understand why a process drift occurred. Further we can imagine that in combination with other analysis methods [15, 19, 20, 22] we can gather a lot more information that helps to identify the root cause for a process drift.

ACKNOWLEDGMENTS

This project (HA project no. 479/15-21) is funded in the framework of Hessen ModellProjekte, financed with funds of LOEWE – Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbund-vorhaben (State Offensive for the Development of Scientific and Economic Excellence) and by the LOEWE initiative (Hessen, Germany) within the NICER project [III L 5-518/81.004].

REFERENCES

[1] Rafael Accorsi and Thomas Stocker. 2012. Discovering Workflow hanges with Time-based Trace Clustering. *Lecture Notes in Business Information Processing* 116 LNBIP (2012), 154–168. DOI: http://dx.doi.org/10.1007/978-3-642-34044-4_9

[2] R P Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Žliobaite, and Mykola Pechenizkiy. 2011. Handling Concept Drift in Process Mining. In *Proceedings of the 23th International Conference on Advanced Information Systems Engineering*. 391–405. DOI: http://dx.doi.org/10.1007/978-3-642-21640-4_30

[3] R. P Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Žliobaite, and Mykola Pechenizkiy. 2014. Dealing With Concept Drifts in Process Mining. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (jan 2014), 154–171. DOI: http://dx.doi.org/10.1109/TNNLS.2013.2278313

[4] Josep Carmona and Ricard Gavaldà. 2012. Online Techniques for Dealing with Concept Drift in Process Mining. In *Jreecs.Com*. 90–102. DOI: http://dx.doi.org/10.1007/978-3-642-34156-4_10

[5] Bill Curtis, Marc I. Kellner, and Jim Over. 1992. Process Modeling. *Commun. ACM* 35, 9 (sep 1992), 75–90. DOI: http://dx.doi.org/10.1145/130994.130998 arXiv:00010782

[6] Christian W. Günther, Stefanie Rinderle, Manfred Reichert, and Wil M. P. van der Aalst. 2006. Change Mining in Adaptive Process Management Systems. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*. Vol. 4275. 309–326. DOI: http://dx.doi.org/10.1007/11914853_19

[7] Shen-Shyang Ho. 2005. A Martingale Framework for Concept Change Detection in Time-Varying Data Streams. *Proceedings of the 22nd International Conference on Machine Learning (ICML-05)* 2004 (2005), 321–327. DOI: http://dx.doi.org/10.1145/1102351.1102392

[8] B F A Hompes, J C A M Buijs, and Wil M. P. van der Aalst. 2015. Detecting Change in Processes Using Comparative Trace Clustering. *Proceedings of the 5th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2015)* (2015), 95–108. http://ceur-ws.org/Vol-1527/paper7.pdf

[9] "IEEE Task Force on Process Mining". 2011. Process Mining Manifesto. *Business Process Management Workshops* (2011), 169–194. DOI: http://dx.doi.org/10.1007/978-3-642-28108-2_19

[10] Geetika T. Lakshmanan, Paul T. Keyser, and Songyun Duan. 2011. Detecting Changes in a Semi-Structured Business Process through Spectral Graph Analysis. In *2011 IEEE 27th International Conference on Data Engineering Workshops*. IEEE, 255–260. DOI: http://dx.doi.org/10.1109/ICDEW.2011.5767640

[11] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. 2014. Process and Deviation Exploration with Inductive visual Miner. *CEUR Workshop Proceedings* 1295 (2014), 46–50.

[12] Abderrahmane Maaradji, Marlon Dumas, and Marcello La Rosa. 2015. Fast and Accurate Business Process Drift Detection. *Lecture Notes in Business Information Processing* 9253 (2015), 406–422. DOI: http://dx.doi.org/10.1007/978-3-319-23063-4

[13] M. V. Manoj Kumar, Likewin Thomas, and B. Annappa. 2015. Capturing the Sudden Concept Drift in Process Mining. *CEUR Workshop Proceedings* 1371, January (2015), 132–143.

[14] John H. McDonald. 2009. Handbook of Biological Statistics. *Sparky House Publishing* (2009), 291. DOI: http://dx.doi.org/10.1017/CBO9781107415324.004 arXiv:arXiv:1011.1669v3

[15] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. 2016. Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders. *Lecture Notes in Artificial Intelligence Series* 9956 LNAI (2016). DOI: http://dx.doi.org/10.1007/978-3-319-46307-0_28

[16] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, Arthur H. M. ter Hofstede, and Boudewijn F. V. van Dongen. 2016. Detecting Drift from Event Streams of Unpredictable Business Processes. 1 (2016), 330–346. DOI: http://dx.doi.org/10.1007/978-3-319-46397-1_26

[17] Manfred Reichert, Clemens Hensinger, and Peter Dadam. 1998. Supporting Adaptive Workflows in Advanced Application Environments. *EDBT Workshop on Workflow Management Systems* (1998), 100–109. http://dbis.eprints.uni-ulm.de/302/

[18] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. 2004. Correctness criteria for dynamic changes in workflow systems - A survey. *Data and Knowledge Engineering* 50, 1 (2004), 9–34. DOI: http://dx.doi.org/10.1016/j.datak.2004.01.002

[19] Alexander Seeliger, Timo Nolle, Benedikt Schmidt, and Max Mühlhäuser. 2016. Process Compliance Checking using Taint Flow Analysis. In *Proceedings of the 37th International Conference on Information Systems - ICIS '16*. Dublin, 1–18.

[20] Suriadi Suriadi, Chun Ouyang, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. 2013. Root Cause Analysis with Enriched Process Logs. *Lecture Notes in Business Information Processing* 132 LNBIP (2013), 174–186. DOI: http://dx.doi.org/10.1007/978-3-642-36285-9-18

[21] Wil M. P. van der Aalst. 2011. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Number 2. Springer Berlin Heidelberg. 352 pages. DOI: http://dx.doi.org/10.1007/978-3-642-19345-3

[22] Evgeniy Vasilyev, Diogo R. Ferreira, and Junichi Iijima. 2013. Using Inductive Reasoning to Find the Cause of Process Delays. In *2013 IEEE 15th Conference on Business Informatics*. IEEE, 242–249. DOI: http://dx.doi.org/10.1109/CBI.2013.41

[23] H. M W Verbeek, Joos C A M Buijs, Boudewijn F. Van Dongen, and Wil M. P. van der Aalst. 2011. XES, XESame, and ProM 6. *Lecture Notes in Business Information Processing* 72 LNBIP (2011), 60–75. DOI: http://dx.doi.org/10.1007/978-3-642-17722-4_5

[24] a. J. M. M. Weijters, Wil M. P. van der Aalst, and a. K. Alves De Medeiros. 2006. Process Mining with the HeuristicsMiner Algorithm. *Cirp Annals-manufacturing Technology* 166 (2006), 1–34. DOI: http://dx.doi.org/10.1.1.118.8288

[25] Michael Werner, Nick Gehrke, and Markus Nuttgens. 2012. Business Process Mining and Reconstruction for Financial Audits. In *2012 45th Hawaii International Conference on System Sciences*. IEEE, 5350–5359. DOI: http://dx.doi.org/10.1109/HICSS.2012.141

[26] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Machine Learning* 23, 1 (1996), 69–101. DOI: http://dx.doi.org/10.1007/BF00116900