Simultaneous Process Drift Detection and Characterization with Pattern-Based Change Detectors

Angelo Impedovo(✉), Paolo Mignone, Corrado Loglisci, and Michelangelo Ceci

Department of Computer Science, University of Bari "Aldo Moro", 70125 Bari, Italy
{angelo.impedovo,paolo.mignone,corrado.loglisci,
michelangelo.ceci}@uniba.it

Abstract. Traditional process mining approaches learn process models assuming that processes are in steady-state. This does not comply with the flexibility and adaptation often requested for information systems and business models. In fact, these approaches should discover variations to adapt to new circumstances, which is a peculiarity that conventional change analysis based on time-series, could not provide, because the processes are complex artifacts. This problem can be handled with change-aware structured representations, such as those typically used for network data. In this paper, we propose a novel pattern-based change detection (PBCD) algorithm for discovering and characterizing changes in event logs encoded as dynamic networks. In particular, PBCDs are unsupervised change detection methods, based on observed changes in sets of patterns observed over time, which are able to simultaneously detect and characterize changes in evolving data. Experimental results, on both real and synthetic data, show the usefulness and the increased accuracy with respect to state-of-the-art solutions.

1 Introduction

The aim of the process mining techniques is learning models (for instance, in the form of Petri nets or heuristic maps) from collections of traces recording observed process executions. Thus, the models can be seen as an abstract form of the really-performed processes and can therefore be used for predictive problems, such as the prediction of outcomes and for conformance checking, that is, the adherence of new traces to the models.

A common assumption of many process mining algorithms is the "invariability" of the process model, meaning that the traces are in a steady-state, that is, they should obey the configuration dictated by the models, without any deviation with respect to the reference process. This aspect has been investigated by methods which recognize variations present in the traces and learn process variants [2]. In many information systems this is not sufficient because the traces might present frequent or regularly repeated changes. A change becomes necessary whenever there is a need for people and institutions to adapt their ordinary behavior to changing circumstances and environments. Various examples

© Springer Nature Switzerland AG 2020
A. Appice et al. (Eds.): DS 2020, LNAI 12323, pp. 451–467, 2020.
https://doi.org/10.1007/978-3-030-61527-7_30

can be found both in society and nature. For example, new regulations and laws require citizens and organizations to change their processes. In a dynamic market, flexible organizations should quickly adapt their internal and external operating procedures to natural disasters as well as to the introduction of new laws and regulations. Therefore, the presence of substantial changes could make the process model inconsistent with respect to the (actual) instances. In order to effectively deal with this, we should revise the working hypothesis and consider the processes as non-stationary, allowing for abrupt or gradual changes exhibited over time. Consequently, the process modeling approaches should react to such process *drifts* by quickly detecting and understanding them [5].

Existing methodologies suffer from several drawbacks. In particular, they work on an over-simplified data representation which does not account for the traces as complex artifacts. This leads to considering only one set of numerical features [4] of the executed traces, while neglecting the temporal component associated to the activities and interactions among the activities, actors and resources, which are sources of information able to explain drifts between traces of the same process model. These representational forms often limit the task of drift detection to a mere quantification of the magnitude of the change between different traces, without providing an explanation of the nature of the change. Thus, any attempt to explain or characterize the changes requires the intervention of the human process modeler or reference knowledge to identify the components of a trace which determine the changes [14].

In this work, we simultaneously solve the problems of process drift detection and characterization with Pattern-based Change Detectors (PBCDs hereafter). PBCDs refer to a class of change detection algorithms in which *i)* the change is detected on patterns discovered from the data over time, and *ii)* the patterns responsible for a given change already constitute an off-the-shelf descriptive model of the change. They have been exploited to study changes on dynamic networks [11] thanks to the peculiarities to identify sub-graphs related to the changes, associate changes to variations of the occurrences of the sub-graphs and quantify the magnitude order of the changes with frequency-based quantitative measures. Thus, our intuition is that of encoding process traces (from the event log) into a graph-based representation and detecting process drifts through PBCDs. This perspective offers several advantages: *i)* the use of an established unsupervised approach to simultaneously solve the problems of drift detection and characterization, *ii)* a computational solution able to account for the temporal order of the activities, *iii)* a method able to determine the most promising set of features mirroring the changes and represent them in form of sub-graph patterns, *iv)* the possibility to capture both gradual changes and sudden changes, which, thus, would appear as mild frequency-based variations and strong frequency-based variations respectively.

The manuscript is organized as follows. Firstly, we introduce some related works in process drift detection and motivate the adoption of PBCDs. Then, we discuss some preliminary notions about processes and dynamic networks, so as to explain how event logs can be transformed into dynamic networks.

The adopted PBCD methodology is then discussed by emphasizing how process drifts are detected and characterized. Then experimental results on both synthetic and real-world event logs are illustrated before drawing some conclusions.

2 Related Works

The adoption of PBCDs for detecting and characterizing the process drifts in event logs occurs at the intersection of two research directions: one concerning pattern-based learning of process models and the other concerning process drift detection methods.

A well-known result in process mining is that *frequent sequential patterns* offer an alternative way of representing process models instead of Petri nets, discovered by the traditional α algorithm [1], or heuristic maps learned by the HeuristicMiner algorithm [17]. Specifically, while *sequential patterns* model the contiguous sequence of executed activities, *frequent sequential patterns* are used to discover statistical evident paths of executions in an event log seen as a database of sequence. Hence, sequential pattern mining algorithms can be used to learn process models as done in [7,8]. An aspect worth mentioning is that frequent patterns effectively model stable features of the process over time. Consequently, our claim is to effectively leverage such features when executing PBCDs on event logs. Unfortunately, to the best of our knowledge, no PBCD based on sequential patterns exists.

As for the process drift detection methods in process mining, different methodologies have been proposed, although none of them is pattern-based. The first is proposed in [4] and implemented in ProM[1], in which the change detection approach is able to detect drifts, via statistical significance testing, by considering a set of four numeric global and local features. In this approach, the event log is transformed into a multivariate time-series, and, hence, changes are detected in such an intermediate representation in which the original control-flow perspective is lost. The second method is the ProDrift algorithm defined in [13] and implemented in the Apromore framework[2]. ProDrift also performs a statistical significance test on a run-based encoding of the traces, obtained prior to the detection phase. Both methods adopt the sliding window model. In particular, the statistical significance test is assessed by comparing the populations of two sliding windows, the reference and the detection windows, that slide over the data whenever a new trace is observed. Both the methods are parametric change detection algorithms, working on an intermediate representation of traces and, lastly, they do not characterize the detected changes.

3 Background

Let A be the set of activities, then an event log over A is defined as the time series of n traces $E = \{T_t\}_{t=1}^{n}$. Each trace $T_i = \langle a_1, \ldots, a_k \rangle$ captures the sequence of k activities $a_i \in A$ as executed at the time point t_i in a given process instance.

[1] https://svn.win.tue.nl/trac/prom/browser/Packages/ConceptDrift.
[2] https://apromore.org/platform/tools/.

Since PBCDs leverage differences between patterns exhibited by the data over time for detecting changes, the principal requirement for their use is the existence of a pattern mining methodology that best suits the data representation at hand. In their natural formulation, event logs are not immediately compatible with traditional pattern mining methods. On the other hand, various existing PBCDs are specifically designed for dynamic networks. Therefore, we encode event logs in the form of dynamic networks as an intermediate representation compatible with existing graph-based PBCDs. Let N be the set of nodes, L be the set of edge labels and $I = N \times N \times L$ the alphabet of all the possible labeled edges. A dynamic network is defined as the time series of n graph snapshots $G = \{G_t\}_{t=1}^{n}$. Each snapshot $G_i \subseteq I$ is a set of edges denoting a directed graph observed in t_i allowing self-loops and multiple edges with different labels.

3.1 From Event Logs to Dynamic Networks

Encoding the event log $E = \{T_t\}_{t=1}^{n}$ as the dynamic network $G = \{G_t\}_{t=1}^{n}$ is done by transforming every trace T_i into the associated graph snapshot $G_i = g(T_i)$. The map $g(T_i)$ allows us to consider the dynamic network $G = \{g(T_t)\}_{t=1}^{n}$ in place of the initial event log. In particular, let $T = \langle a_1, \ldots, a_k \rangle$ be a trace, the graph $G = g(T)$ is built by considering i) the set of edge labels $L = \{a_1, \ldots, a_k\}$, ii) the set of nodes $N = \{0, 1, \ldots, n\}$ and iii) $I = N \times N \times L$. Then, $G = g(T) = \{(i-1, i, a_i) \in I \mid a_i \in T\} \subseteq I$ is a labeled graph in which edge labels denote activities, and nodes denote natural numbers. This graph-based representation of traces keeps the temporal ordering of activities in a trace, as shown in Fig. 1, and this is a necessary condition to preserve the process control-flow perspective in the drift detection activity.

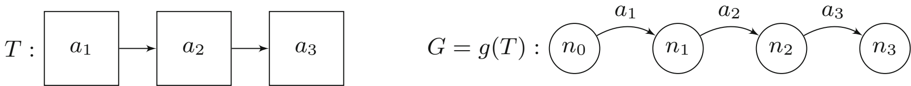

Fig. 1. Example of a trace T made of 3 activities (a_1, a_2 and a_3) represented as the graph snapshot $G = g(T)$. Activity names in T become edge labels in G.

3.2 Frequent and Emerging Subgraph Discovery

The representation of event logs as dynamic networks fits the one adopted in transactional data mining, meaning that it is possible to discover interesting sub-graphs with traditional sub-graph mining algorithms designed for dynamic networks. In the transactional setting, a snapshot $G_{tid} \in G$ is a transaction uniquely identified by *tid*, whose items are labeled edges from I. A sub-graph $S \subseteq I$, with *length* $|S|$, can be seen as a word $S = \langle i_1 \ldots i_n \rangle$ of n lexicographic sorted items, with prefix $P = \langle i_1 \ldots i_{n-1} \rangle$ and suffix i_n.

For this work *frequent connected sub-graphs* (FCSs hereafter) are deemed to be interesting, as they denote stable features that are useful for the drift detection step. FCSs are discovered from graph snapshots belonging to time windows. A window $W = [t_i, t_j]$, with $t_i < t_j$, is the sequence of snapshots $\{G_i, \ldots, G_j\} \subseteq G$. Consequently, the width $|W| = j - i + 1$ is equal to the number of snapshots collected in W. Let S be a sub-graph, then the *tidset* of S in the window W is defined as $tidset(S, W) = \{tid \mid \exists G_{tid} \in W \wedge S \subseteq G_{tid}\}$, while the *support* of S in W is $sup(S, W) = \frac{|tidset(S,W)|}{|W|}$. S is *frequent* in W if $sup(S, W) > minSUP$, where $minSUP \in [0, 1]$. We term F_W the set of all the FCSs in the window W.

Once detected a process drift needs to be characterized. While the FCSs support the drift detection by capturing statistically evident parts of the process, as observed in a time window, they do not characterize drifts. To this end, we deem interesting the *emerging connected sub-graphs* (ESs hereafter), discovered between two time windows by evaluating the growth-rate of sub-graphs. Let S be a sub-graph, W and W' two consecutive time windows, then the *growth-rate* of S between W and W' is $gr(S, W, W') = \frac{max(sup(S,W), sup(S,W'))}{min(sup(S,W), sup(S,W'))}$. S is *emerging* between W and W' if $gr(S, W, W') > minGR$, where $minGR > 1$. We term $es(W, W')$ the set of the ESs between W and W' according to $minGR$.

The ESs are the building blocks of the change characterizations. However, i) the combinatorial explosion of the ESs worsens the readability of characterizations, and ii) ESs singularly add small contributions to the characterizations. To tackle these problems, we only consider the *maximal emerging connected sub-graphs* (MESs hereafter). Let $S \in es(W, W')$, then S is maximal if there is not another sub-graph $Q \in es(W, W')$ such that $S \subset Q$. We term $ms(W, W')$ the set of all the MESs between W and W' according to $minGR$.

3.3 Problem Statement

Let $E = \{T_t\}_{t=1}^{n}$ be an event log, $minSUP \in [0, 1]$ be the minimum support threshold, $minMC \in [0, 1]$ the minimum change threshold, $minGR > 1$ be the minimum growth-rate threshold. Then:

- the *dynamic network* $G = \{g(T_t)\}_{t=1}^{n}$ of E is built as a pre-processing step.
- *pattern-based change detection* finds pairs of windows $W = [t_b, t_e]$ and $W' = [t_{e+1}, t_c]$ from D, where $t_b \leq t_e < t_{e+1} \leq t_c$. Each pair of windows denotes a change which is:
  - quantified by the pattern dissimilarity score $d(F_W, F_{W'}) > minMC$
  - explained by the maximal emerging sub-graphs $ms(F_W, F_{W'})$ discovered according to $minGR$

where F_W ($F_{W'}$) denote the FCSs discovered on W (W') according to $minSUP$.

Process drifts are detected on the dynamic network encoding of the event log. Specifically, a drift is detected every time a relevant difference between the set of FCSs F_W and $F_{W'}$ is measured. Finally, the drift is explained by the MESs, as they describe the (appearing or disappearing) sequences of activities involved in the change of the underlying process model.

3.4 Computational Approach

The afore-mentioned change detection and explanation problem can be solved by various computational solutions. Among them, we mention the class of pattern-based change detection algorithms (PBCD). In general, a PBCD forms a two-step approach in which: i) a pattern mining algorithm extracts the set of patterns observed from the incoming data, and ii) the amount of change is quantified by adopting a dissimilarity measure defined between sets of patterns. More specifically, a PBCD is an iterative algorithm that consumes data coming from a data source, in our case a dynamic network, and produces quantitative measures of changes. For instance, the KARMA algorithm proposed in [11] is a PBCD for detecting and characterizing changes in network data. KARMA is based on the exhaustive mining of FCSs, whose general workflow can be seen in Figure 2. The algorithm iteratively consumes blocks Π of graph snapshots coming from D (Step 2) by using two successive landmark windows W and W' (Step 3). Thus, it mines the complete sets of FCSs, F_W and $F_{W'}$, which are necessary for the detection steps (Steps 4–5). The window grows ($W = W'$) with new graph snapshots, and the associated set of FCSs is kept updated (Steps 8–9) until the change score $d(F_W, F_{W'})$ exceeds β and a change is detected. In that case, the algorithm drops the content of the window by retaining only the last block of transactions ($W = \Pi$, Steps 6–7). Then the analysis restarts.

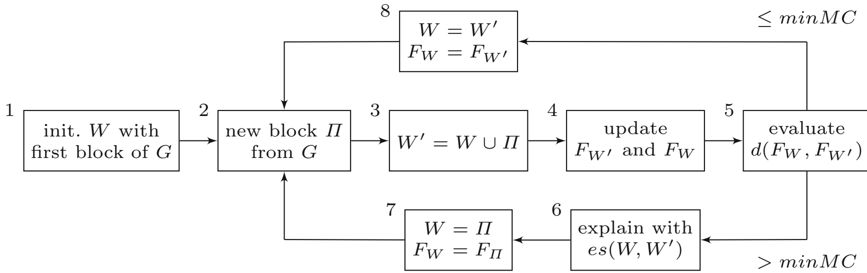

Fig. 2. The KARMA algorithm flowchart

However, KARMA does not naturally fit the given problem statement and is not the optimal solution. Firstly, while KARMA relies on successive landmark windows of increasing size, our problem statement compares two successive non-overlapping windows of different size. Secondly, KARMA discovers FCSs on data represented in a more general representation than the dynamic network encoding of event logs in the form of sequences of graph chains. Consequently, no FCS, which is not a simple chain, would be returned by the mining algorithm: chains are only discovered when mining FCSs in sequences of chains. However, although this solution is always able to discover FCSs that are also chains, it is also inefficient: the mining algorithm would also generate and discard the FCSs which are not chains. Therefore, we restrict the pattern language to *frequent subtrees*

(FSs hereafter), that is, FCSs in which every node is connected to a parent node, except for the root node. To meet these requirements, we adapt the KARMA algorithm to the KARMATree approach depicted in Fig. 3. In this case, an alternative time window model is used to arrange incoming blocks of transactions (Steps 3, 7 and 8). Then sets F_W and $F_{W'}$ of FSs are discovered instead of FCSs (Step 4). Lastly, changes are characterized by discovering the maximal emerging subtrees (Step 6) instead of the emerging ESs by KARMA.

Let G be a dynamic network over $|I| = k$ possible edges, with n snapshots and $m = \frac{n}{|\Pi|}$ blocks of size $|\Pi|$. KARMA requires time proportional to $O(m \cdot |FCSs|)$ in the worst case scenario [11], while KARMATree requires $O(m \cdot |FSs|)$ where $|FSs| << |FCSs| < e^k$, since the number of subtrees is lower than the number of subgraphs in a network. However, KARMATree is an exhaustive PBCD, relying on complete mining of FSs, which could not work well in limited memory scenarios. As a solution, a non-exhaustive variant could be obtained by equipping KARMATree with the heuristic mining approach shown in [10].

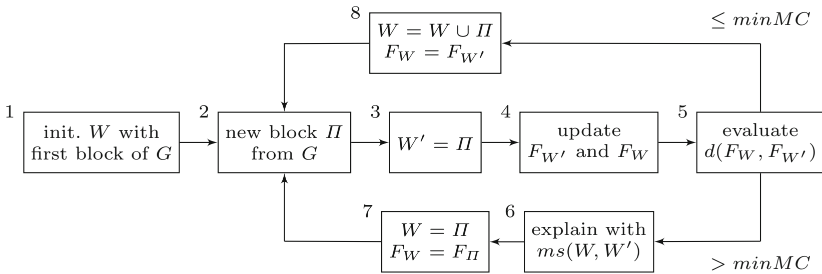

Fig. 3. The KARMATree algorithm flowchart

4 Experiments

The experiments are organized according to different perspectives concerning both synthetic and real-world processes. In particular, we answer the following research questions: **Q1)** Is the proposed PBCD approach *more accurate* than existing process drift detection approaches when detecting changes on synthetic processes? **Q2)** Is the proposed PBCD approach *more efficient* than existing process drift detection approaches when detecting changes on synthetic processes? **Q3)** Do the characterizations describe changes in real-world process?

Assessing the accuracy of the proposed approach compared with competitor methodologies is not an easy task. Although the process evolution is a well-established concept in process mining, to the best of our knowledge, no proper ground truth for process drift detection is known. The main consequence is the difficulty of measuring the accuracy on real-world datasets. Moreover, existing synthetic log generators are not flexible enough. For instance, the one proposed

in [6] randomly builds and evolves single process models and simulates their execution to synthesize event logs.

The major limitation is that the simulation is based only on a single process model, and hence none of its evolutions is considered. This means that i) traces in the resulting event log conform to the process model used in the simulation, and ii) consequently no evident change is injected into the resulting log. To overcome this limitation, we extended the process log generator[3] to i) build a chain of n process models where the first is randomly generated and the others are subsequent random evolutions, and ii) generate the complete event log by simulating an equal number of traces for every process model in the chain. We synthesized 10 event logs, each built by considering a chain made of 20 evolving process models. In particular, each model has been used to simulate a block made of 100 traces, for a total number of 2,000 traces per each log, thus ensuring a change between every pair of subsequent blocks.

By so doing, every generated log can be used as a ground-truth about the presence of changes when evaluating the accuracy of the proposed PBCD approach.

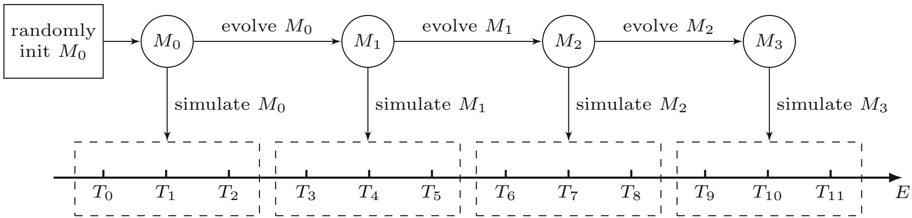

Fig. 4. Synthetic event log E built by simulating the execution of a process model M_0 which evolves 3 times. Each model evolution produces a block of 3 traces.

To answer the afore-mentioned research questions, we first discuss the results of a comparative evaluation between our approach and existing drift detectors for process data. Then a case study is shown to illustrate the usefulness of KARMATree for simultaneously detecting and characterizing changes in real world datasets. In particular we compare our proposed KARMATree PBCD approach with two state-of-the-art process drift detection algorithms, the *ProDrift* [13] available in the Apromore framework and the drift detector by Bose et al. [4] available in the ProM framework, respectively. Both the competitors are parametric change detection algorithms specifically designed for process data. They are built so as to embed the ADWIN [3] algorithm, therefore, they discover changes in event logs by scanning them through adaptively-sized time windows (we term these two algorithms *ProDrift (adwin)* and *Bose et al. (adwin)*). The *ProDrift* algorithm can also be used with fixed-size windows, termed as *ProDrift (fixed)*. Another difference between KARMATree and both ProDrift

[3] https://bitbucket.org/carbonkid/process/.

and Bose et al. is their detection method. In fact, while KARMATree is a non-parametric drift detection approach, relying on pattern-set dissimilarities, both ProDrift and Bose et al. seek changes by performing statistical hypothesis testing, at a given p-value, between the data population in the time windows. Specifically, Bose et al. employ two global features which are defined over an event log, and two local features, which are defined at a trace level by considering a fixed-size window, while ProDrift works only at trace level by considering an event log as a continuous stream of traces and it is designed to adaptively identify the right window size.

4.1 The Most Accurate Process Drift Detection Approach

In this set of experiments we executed KARMATree and the three competitor algorithms on 10 synthetic event logs, generated according to the procedure previously described, and collected their accuracies, false positive rates (FPRs) and false negative rates (FNRs). We fixed the initial size of time windows to 20 in every considered approach, as for KARMATree, we fixed the minimum support threshold to $minSUP = 0.1$ and the minimum growth-rate threshold to $minGR = 1.0$. On the contrary, we tuned the minimum change threshold as $minMC = \{0.5, 0.6, 0.7, 0.8, 0.9\}$. On the other hand, we fixed a critical p-value of 0.95 for both ProDrift and Bose et al.

Table 1. Accuracy of KARMATree against ProDrift (fixed, adwin) and Bose et al. (adwin) when tuning *minMC* on 10 synthetic event logs.

dataset	Accuracy @ *minMC*							
	KARMATree					ProDrift	ProDrift	Bose et al.
	0.5	0.6	0.7	0.8	0.9	(fixed)	(adwin)	(adwin)
synth-log-01	0.989	0.989	0.989	0.959	0.918	1	0.846	0.959
synth-log-02	0.959	0.959	0.948	0.928	0.867	0.989	0.877	0.858
synth-log-03	1	1	1	0.979	0.959	1	0.816	0.909
synth-log-04	1	1	0.989	0.938	0.857	1	0.846	0.898
synth-log-05	0.959	0.969	0.959	0.948	0.918	0.959	0.857	0.929
synth-log-06	0.989	0.979	0.959	0.908	0.867	0.928	0.846	0.898
synth-log-07	0.989	0.979	0.979	0.938	0.908	1	0.816	0.929
synth-log-08	0.969	0.969	0.948	0.928	0.887	1	0.846	0.939
synth-log-09	0.979	0.969	0.928	0.908	0.857	1	0.826	0.959
synth-log-10	0.969	0.969	0.969	0.959	0.938	1	0.826	0.939

As for the accuracy (Table 1), we report that KARMATree always outperforms ProDrift (adwin) for every value of *minMC*. The same is not true for Bose et al. (adwin), which is outperformed by KARMATree when $minMC \leq 0.8$, and outperforms KARMATree when $minMC = 0.9$. On the contrary, ProDrift (fixed) is a top competitor based on time windows of fixed size, differently from KARMATree and every adwin-based competitor adopting time windows of dynamic size. Specifically, since i) ProDrift (fixed) consumes blocks of 20 traces, and ii) the synthetic datasets are generated so to report a change once every 100 traces, the algorithm compares two clearly distinct group of traces once every 5 windows, on which a change is detected. However, knowing in advance the temporal distribution of changes (once every 100 traces) requires prior knowledge on the observed process, which could not always be available. In this perspective, differently from the remaining adwin-based competitors, KARMATree still outperforms ProDrift (fixed) on synth-log-03/05/06 (for $minMC \leq 0.7$), and synth-log-04 (for $minMC \leq 0.6$).

This analysis is confirmed by the false positive rates (FPRs) and false negative rates (FNRs). As expected, ProDrift (fixed) exhibits both FPRs and FNRs approximately equal to 0 on every dataset. Also, ProDrift (adwin) exhibits very low FPRs but moderately high FNRs. On the contrary Bose et al. exhibits the worst FPR on almost every dataset (except for synth-log-09) and remarkable FNRs, which in turn are no worse than ProDrift (adwin). As for KARMATree, the algorithm always outperforms the competitors with respect to their FPRs. As for the FNRs, KARMATree outperforms every competitor for low values of *minMC*. From these results, two tendencies arise: i) both FPRs and FNRs decrease with *minMC*, and ii) the accuracy increases for low values of *minMC* (Table 2).

4.2 The Most Efficient Process Drift Detection Approach

In this set of experiments we compared the running times (seconds) of KARMATree against the ones of the three competitors on 10 synthetic event logs (Table 3). As before, we fixed the initial size of time windows to 20 in every considered approach. As for KARMATree we fixed the minimum support to $minSUP = 0.1$ and the minimum growth-rate to $minGR = 1.0$. On the contrary, we tuned the minimum change threshold as $minMC = \{0.5, 0.6, 0.7, 0.8, 0.9\}$. We fixed a critical p-value of 0.95 for ProDrift and Bose et al. No clear tendency emerges when looking at decreasing values of *minMC* for KARMATree. This is an expected result, since *minMC* does not influence the running times, which are strongly determined by the mining step in the PBCD pipeline. However, when comparing KARMATree with respect to both ProDrift fixed and adwin-based,

Table 2. False positive rate (FPR) and False negative rate (FNR) of KARMATree against ProDrift (fixed, adwin) and Bose et al. (adwin) when tuning *minMC* on 10 synthetic event logs.

dataset	False positive rate @ *minMC*							
	KARMATree					ProDrift	ProDrift	Bose et al.
	0.5	0.6	0.7	0.8	0.9	(fixed)	(adwin)	(adwin)
synth-log-01	0	0	0	0	0	0	0	0
synth-log-02	0	0	0	0	0	0.013	0	0.087
synth-log-03	0	0	0	0	0	0	0.013	0.062
synth-log-04	0	0	0	0	0	0	0	0.087
synth-log-05	0.013	0	0	0	0	0.051	0.013	0.0375
synth-log-06	0	0	0	0	0	0.051	0.063	0.075
synth-log-07	0	0	0	0	0	0	0	0
synth-log-08	0	0	0	0	0	0	0	0.05
synth-log-09	0.013	0	0	0	0	0	0	0
synth-log-10	0	0	0	0	0	0	0	0
dataset	False negative rate @ *minMC*							
	KARMATree					ProDrift	ProDrift	Bose et al.
	0.5	0.6	0.7	0.8	0.9	(fixed)	(adwin)	(adwin)
synth-log-01	0.053	0.053	0.053	0.211	0.421	0	0.789	0.211
synth-log-02	0.211	0.211	0.263	0.368	0.684	0	0.632	0.368
synth-log-03	0	0	0	0.105	0.211	0	0.895	0.211
synth-log-04	0	0	0.053	0.316	0.737	0	0.789	0.158
synth-log-05	0.158	0.158	0.211	0.263	0.421	0	0.684	0.211
synth-log-06	0.053	0.105	0.211	0.474	0.684	0.158	0.526	0.211
synth-log-07	0.053	0.105	0.105	0.316	0.474	0	0.947	0.368
synth-log-08	0.158	0.158	0.263	0.368	0.579	0	0.789	0.105
synth-log-09	0.053	0.158	0.368	0.474	0.737	0	0.895	0.211
synth-log-10	0.158	0.158	0.158	0.211	0.316	0	0.895	0.316

our approach is more efficient than the two competitors (except for synth-log-03 when $minMC \leq 0.7$). Moreover, KARMATree is more efficient than Bose et al. by at most two orders of magnitude. Therefore, we conclude that KARMATree is able to devise more accurate and more efficient drift detection on almost every considered dataset.

Table 3. Running times (seconds) of KARMATree against ProDrift (fixed, adwin) and Bose et al. (adwin) when tuning *minMC* on 10 synthetic event logs.

dataset	Running times (seconds) @ *minMC*							
	KARMATree					ProDrift	ProDrift	Bose et al.
	0.5	0.6	0.7	0.8	0.9	(Fixed)	(Adwin)	(Adwin)
synth-log-01	0.86	0.874	0.891	1.106	0.856	3.38	3.453	28.284
synth-log-02	1.499	1.625	1.688	1.782	1.58	3.908	3.411	47.411
synth-log-03	6.107	6.155	5.932	4.966	5.735	5.005	5.054	187.65
synth-log-04	3.188	3.257	3.392	3.203	3.705	4.932	4.753	92.523
synth-log-05	1.471	1.452	1.592	1.532	1.437	4.172	3.558	47.311
synth-log-06	0.658	0.628	0.629	0.642	0.606	2.79	2.795	18.789
synth-log-07	3.047	2.764	2.954	3.066	3.065	4.4	4.251	96.541
synth-log-08	2.489	2.496	2.717	2.584	2.562	4.777	4.449	70.165
synth-log-09	3.85	3.504	4.43	4.353	5.105	6.243	5.421	101.7
synth-log-10	1.436	1.375	1.445	1.512	1.367	3.83	3.714	42.003

4.3 Case Study

We illustrate a case study in which KARMATree is used to detect and characterize the changes in a real process. When a change is detected between two windows, the reference window and the target window, it is reasonable to expect some differences between the two associated process models. Intuitively, the change can be characterized by listing the modifications necessary to transform the process model, learned from traces in the reference window, into the one learned from traces in the target window. We recall that KARMATree characterizes changes by listing the maximal emerging subtrees between the reference and the target windows. Two considerations arise: first, it is possible that a subtree which was frequent in the reference window becomes infrequent in the target window, and second, a subtree which was infrequent in the reference window may become frequent in the target window. Since an emerging subtree denotes an appearing (disappearing) sequence of activities, then the associated activities will (will not) be included in the process models. Furthermore, since KARMATree discovers *maximal emerging subtrees*, the change is characterized in terms of the longest sequences of activities which appear or disappear over time.

The real process we consider is the hospital billing process collected in [15]. The event log collects events related to the billing of medical services as provided by a regional hospital. The dataset was collected from the financial modules of the ERP system of the hospital. Specifically, the event log contains 100.000 anonymized traces recorded over a period of three years. Our purpose is to detect and characterize the changes in the hospital billing process. With this objective in mind, we first encoded the dataset as a dynamic network by following the procedure described in Sect. 3. Then we executed the KARMATree algorithm

on the resulting dynamic network by fixing the input parameters as follows: the minimum support threshold was fixed at $minSUP = 0.1$, the minimum change threshold at $minMC = 0.5$ and the minimum growth-rate threshold at $minGR = 1.0$. Once it had been executed, the algorithm was able to detect and characterize 95 change points out of 1000 blocks.

We report two changes detected by KARMATree, by focusing on the associated characterizations. In particular, given a change detected between a reference window W and a target window W', we match the MESs against the two heuristic maps discovered by running the HeuristicMiner algorithm (available in ProM [17]) on traces from the two windows, respectively. Thus, we show how MESs, discovered by KARMATree, mirror the differences between the two heuristic maps. We note that heuristic maps highlight the frequently executed parts of the process in black, while the less executed ones appear in gray.

The first change is detected when KARMATree consumes 6000 traces. Specifically, the change score amounts to 54% and is spotted between the reference window containing the first 5900 traces ($W = [1, 5900]$) and the target window containing the remaining 100 ($W' = [5901, 6000]$). The billing process in W is depicted in Fig. 5. First, a new billing is created and the associated diagnosis is set. Then, the fine is created and released. Consequently, the bill is closed only when the fine is released with success. Occasionally, the diagnosis can be changed multiple times before deleting the billing prematurely (due to errors, for example). Errors can also affect the fine, in that case a new one is created by returning to the create fine activity. However, the billing process looks different when observed in W' (Fig. 6). The change diagnosis activity is less frequently executed than in W, and may cause the deletion of the bill. Consequently, a new fine is created right after the billing process starts. When comparing this heuristic map to the previous one, it emerges that the billing process has been shortened by avoiding the change diagnosis activity.

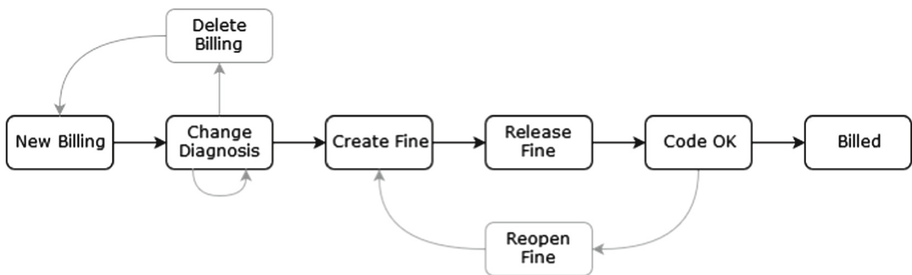

Fig. 5. Heuristic map for the billing process in the *reference window* $W = [1, 5900]$.

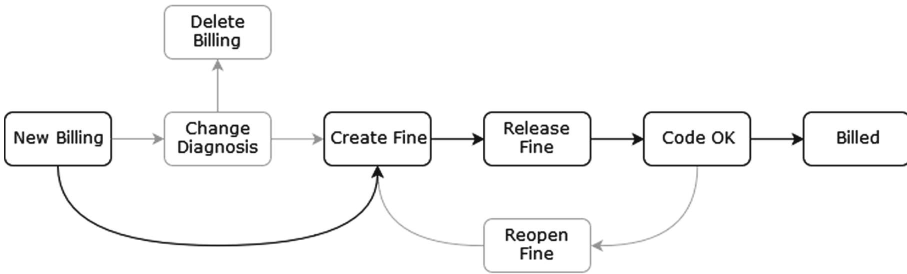

Fig. 6. Heuristic map for the billing process in the *target window* $W' = [5901, 6000]$.

KARMATree explains the change with a single *maximal emerging pattern* S which is frequent in W, with a support of 39%, and infrequent in W' with a support of 7%. Therefore, S emerges with a growth-rate of 558%:

$$S = \{(0, 1, \text{new billing}), (1, 2, \text{change diagnosis}), (2, 3, \text{create fine}),$$
$$(3, 4, \text{release fine}), (4, 5, \text{code ok}), (5, 6, \text{billed})\}$$

Since S is emerging and infrequent in W', it suggests that the billing process, as performed in W, is not compliant with the process model observed on W'. This is an expected result, since the heuristic map discovered on W' does not depict the billing process as S does. On the contrary, S is compliant with the heuristic map discovered on W. Clearly, every subtree $S' \subset S$ involving the change diagnosis activity also characterizes the change occurring between W and W' (for example, $S' = \{(0, 1, \textit{new billing}), (1, 2, \textit{change diagnosis}), (2, 3, \textit{create fine})\}$). Indeed, the same change could have been represented by various emerging subtrees, each of which adds a small contribution to the remaining ones. The usefulness of discovering *maximal emerging subtrees* is precisely the characterizing of changes in a succinct way, that is, by only considering the longest sequences of activities.

KARMATree detects a second change (54%) immediately after the arrival of 100 new traces. In this case, the reference window is $W = [5901, 6000]$ and is equivalent to the target window of the previous example, while the current target window is $W' = [6001, 6100]$. Consequently, the heuristic map associated to W is depicted in Fig. 6, and the one associated to W' is depicted in Fig. 7. This new heuristic map specifies that the billing process can be alternatively completed by either changing the diagnosis associated to the billing or not. Moreover, the map also states that i) new billings have been immediately deleted after their creation and ii) no fine is reopened. The change is characterized by two maximal emerging subtrees S_1 and S_2 which are infrequent in W (support of 7% and 4%, resp.) and frequent in W' (support of 42% and 12%, resp.).

$$S_1 = \{(0, 1, \text{new billing}), (1, 2, \text{change diagnosis}), (2, 3, \text{create fine}),$$
$$(3, 4, \text{release fine}), (4, 5, \text{code ok}), (5, 6, \text{billed})\}$$
$$S_2 = \{(0, 1, \text{new}), (1, 2, \text{delete})\}$$

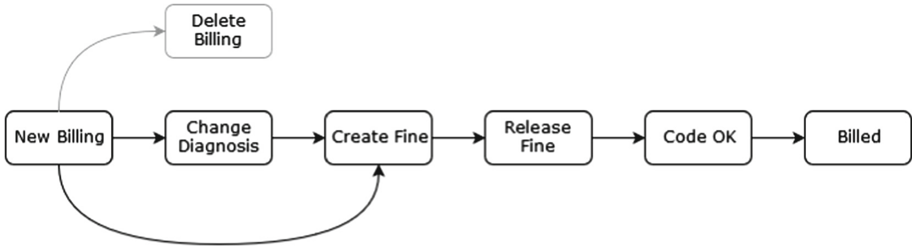

Fig. 7. Heuristic map for the billing process in the *reference window* $W = [6001, 6100]$.

Since both the subtrees are frequent and emerging in W', they denote novel parts of the process, as executed according to the heuristic map on W'. In particular, while S_1 reintroduces the change diagnosis activity in the billing process, S_2 states that new billings are immediately deleted right after their creation. We note that the two subtrees are compliant with the heuristic map learned on W'.

5 Conclusions

We have presented the KARMATree for simultaneously detecting and characterizing process drifts. KARMATree detects changes in an intermediate representation of event logs in the form of dynamic networks. Specifically, changes are i) sought by tracking variations in the frequent subtrees observed over time on non-overlapping time windows and ii) characterized with maximal emerging subtrees. Experiments have shown that KARMATree is more efficient and more accurate than existing state-of-the-art process drift detection algorithms. Furthermore, a case study on real world data has shown that the characterizations provided by KARMATree spot parts of the process involved in a given change. As to future research directions, we plan to i) improve the efficiency through the use of filter-and-refinement techniques, already explored on spatio-temporal data [16], ii) work on the conciseness of the changes through condensed representations of the patterns [9], iii) study the process drift over a longer temporal horizon through evolution chains [12].

Acknowledgments. We acknowledge the support of the MIUR - Ministero dell'Istruzione dell'Universitàe della Ricerca through the project "TALIsMan - Tecnologie di Assistenza personALizzata per il Miglioramento della quAlità della vitA" (Grant ID: ARS01_01116), funding scheme PON RI 2014–2020. We would also like to thank Lynn Rudd for her help in reading the manuscript.

References

1. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004). https://doi.org/10.1109/TKDE.2004.47
2. Assy, N., van Dongen, B.F., van der Aalst, W.M.P.: Discovering hierarchical consolidated models from process families. Adv. Inf. Syst. Eng. - CAiSE **2017**, 314–329 (2017). https://doi.org/10.1007/978-3-319-59536-8_20
3. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the Seventh SIAM International Conference on Data Mining, pp. 443–448 (2007).https://doi.org/10.1137/1.9781611972771.42
4. Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Handling concept drift in process mining. In: Advances Information Systems Engineering, pp. 391–405 (2011). https://doi.org/10.1007/978-3-642-21640-4_30
5. Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Networks Learn. Syst. **25**(1), 154–171 (2014). https://doi.org/10.1109/TNNLS.2013.2278313
6. Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. BPM Demo Track **2016**, 1–6 (2016)
7. Ceci, M., Lanotte, P.F., Fumarola, F., Cavallo, D.P., Malerba, D.: Completion time and next activity prediction of processes using sequential pattern mining. In: Discovery Science - 17th International Conference, pp. 49–61 (2014). https://doi.org/10.1007/978-3-319-11812-3_5
8. Hassani, M., Siccha, S., Richter, F., Seidl, T.: Efficient process discovery from event streams using sequential pattern mining. In: IEEE Symposium on Computer Intelligence 2015, pp. 1366–1373 (2015). https://doi.org/10.1109/SSCI.2015.195
9. Impedovo, A., Loglisci, C., Ceci, M., Malerba, D.: Condensed representations of changes in dynamic graphs through emerging subgraph mining. Eng. Appl. Artif. Intell. **94** (2020). https://doi.org/10.1016/j.engappai.2020.103830
10. Impedovo, A., Ceci, M., Calders, T.: Efficient and accurate non-exhaustive pattern-based change detection in dynamic networks. In: Discovery Science - 22nd International Conference, DS 2019, Split, Croatia, 28–30 October 2019, Proceedings, pp. 396–411 (2019). https://doi.org/10.1007/978-3-030-33778-0_30
11. Loglisci, C., Ceci, M., Impedovo, A., Malerba, D.: Mining microscopic and macroscopic changes in network data streams. Knowl. Based Syst. **161**, 294–312 (2018)
12. Loglisci, C., Ceci, M., Malerba, D.: Discovering evolution chains in dynamic networks. In: New Frontiers in Mining Complex Patterns - First International Workshop, NFMCP 2012, Held in Conjunction with ECML/PKDD 2012, Bristol, UK, 24 September 2012, Revised Selected Papers, pp. 185–199 (2012). https://doi.org/10.1007/978-3-642-37382-4_13
13. Maaradji, A., Dumas, M., Rosa, M.L., Ostovar, A.: Fast and accurate business process drift detection. In: Business Process Management - 13th International Conference, pp. 406–422 (2015). https://doi.org/10.1007/978-3-319-23063-4_27
14. Maaradji, A., Dumas, M., Rosa, M.L., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE Trans. Knowl. Data Eng. **29**(10), 2140–2154 (2017). https://doi.org/10.1109/TKDE.2017.2720601
15. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Data-driven process discovery - revealing conditional infrequent behavior from event logs. In: Advances Information Systems Engineering - 29th International Conference, pp. 545–560 (2017). https://doi.org/10.1007/978-3-319-59536-8_34

16. Vieira, M.R., Bakalov, P., Tsotras, V.J.: On-line discovery of flock patterns in spatio-temporal data. In: 17th ACM International Symposium on Advances in Geographic Information Systems, pp. 286–295 (2009). https://doi.org/10.1145/1653771.1653812
17. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, France, pp. 310–317 (2011). https://doi.org/10.1109/CIDM.2011.5949453