Online Techniques for Dealing with Concept Drift in Process Mining

Josep Carmona and Ricard Gavaldà

Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract. *Concept drift* is an important concern for any data analysis scenario involving temporally ordered data. In the last decade *Process mining* arose as a discipline that uses the *logs* of *information systems* in order to mine, analyze and enhance the process dimension. There is very little work dealing with concept drift in process mining. In this paper we present the first online mechanism for detecting and managing concept drift, which is based on *abstract interpretation* and *sequential sampling*, together with recent learning techniques on data streams.

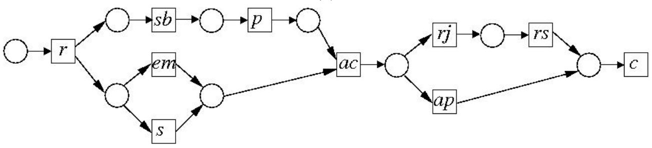
1 Introduction

Process Mining is a relatively novel discipline which has received a lot of attention in the last decade [16]. Although it shares many features with Data Mining, it has originated from different concerns and communities, has a set of distinctive techniques, and produces slightly different outcomes. Historically, process mining arises from the observation that many organizations record their activities into *logs* which describe, among others, the real ordering of activities of a given process, in a particular implementation. *Software engineering* techniques have mainly focused on the specification part of the processes within an *information system.* In reality, this may cause a big gap between a system specification's and the final implementation, hampering the use of the models that specify the main processes of an information system. As another example, designers of hardware or embedded, concurrent systems, need to compare behavior and specifications; typically they can passively or actively generate large amounts of logs from their target system and/or their prototypes, so a logical approach is to use these logs for the verification task.

By using the logs as source of information, process mining techniques are meant to *discover*, *analyze,* and *enhance* formal process models of an information system [17]. Process discovery is probably the main and most challenging discipline within process mining: to discover a formal process model (a Petri net [15], an automaton, etc.) that adequately represents the traces in the log. Several process discovery algorithms exist in the literature (the reader can find a good summary in [17]). In this paper, we concentrate on the *control-flow* part, i.e., the causal relations between the events of a process. Let us use an example to illustrate control-flow discovery. The example shown in Figure 1 is taken from [18] and considers

1	r,s,sb,p,ac,ap,c	8	r,em,s,sb,p,ac,ap,c
2	r,sb,em,p,ac,ap,c	9	r,sb,em,s,p,ac,ap,c
3	r,sb,p,em,ac,rj,rs,c	10	r,sb,em,s,p,ac,rj,rs,c
4	r,em,sb,p,ac,ap,c	11	r,em,sb,p,s,ac,ap,c
5	r,sb,s,p,ac,rj,rs,c	12	r,em,sb,s,p,ac,rj,rs,c
6	r,sb,p,s,ac,ap,c	13	r,em,sb,p,s,ac,ap,c
7	r,sb,p,em,ac,ap,c	14	r,sb,p,em,s,ac,ap,c

(a)



(b)

Fig. 1. Control-flow process discovery: (a) log containing a drift from trace 8 on, (b) Petri net discovered from part of the log (traces 1 to 7)

the process of handling customer orders. In the example, the log contains the following activities: *r=register*, *s=ship*, *sb=send_bill*, *p=payment*, *ac=accounting*, *ap=approved*, *c=close*, *em=express_mail*, *rj=rejected*, and *rs=resolve*. The goal of process discovery is to obtain a formal model such as the Petri net shown in Figure 1(b)[1].

The problem of *concept drift* is well known and well studied in the data mining and machine learning communities, but hardly addressed so far in process mining, where it has important particularities. Three main problems regarding concept drift can be identified in the context of process mining [5]:

1. *Change Detection*: detect when a process change happens. This is the most fundamental problem to solve.
2. *Change Localization and Characterization*: characterize the nature of one particular change, and identify the region(s) of change in a process.
3. *Unraveling Process Evolution*: discover the evolution of process change overall, and how change affects the model over time.

Current process discovery algorithms behave poorly when a log incorporates a drift: causal relations between events may appear and disappear, or even reverse, and therefore cannot be resolved. For instance, in the example of Figure 1, in the first part of the log (traces 1–7) activities *em* and *s* are in conflict, i.e., only one of them can be observed in a process execution. However, from trace 8 on, both activities occur with *em* always preceding *s*. Thus, the whole log contains both behaviors and therefore process discovery techniques fail at determining the causal relationship between *em* and *s*. In that case, no arcs connecting activities *s* and *em* with the rest of activities in the model are discovered.

[1] For the reader not familiar with Petri nets: a transition (box) is enabled if every input place (circle) holds a token (black dot). If enabled, the transition can fire, removing tokens from its input places and adding tokens to its output places.

This paper presents an online technique to detect concept drift by sequential monitoring of the logs of a system. It is a multi-stage technique that uses the theory of *abstract interpretation* [10] to learn an internal representation (in terms of a *polyhedron* or an *octagon*), that is afterwards used to estimate the faithfulness of the representation in including the traces in the log. For the estimation and concept drift detection, we use an adaptive window technique [1] which has been proved to be very effective for similar purposes [2–4]. Remarkably, the techniques presented in this paper are automatic by nature, e.g., no user-intervention is required.

To our knowledge, the only work in the literature that addresses concept drift in process mining is [5], where *statistical hypothesis testing* is applied to detect *post-mortem* the drifts present in a log. Our approach, by contrast, is intended detect and react to changes in an online, almost real-time way. The technique may be used in different scenarios, such as: (i) for an *a posteriori* analysis, as in [5]; ii) as a preprocessor in an online setting, to segment the log and apply process discovery techniques separately to drift-free segments; (iii) to monitor a system in order to detect deviations from the expected behavior (embodied in an existing model).

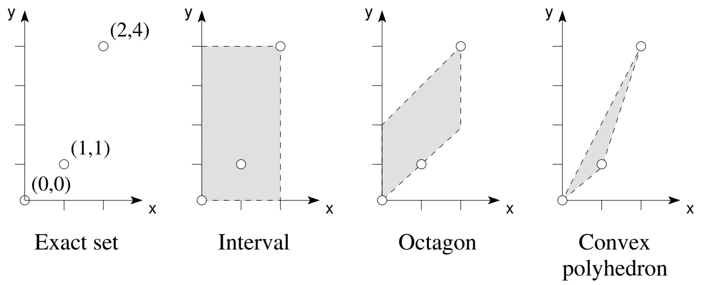
2 Background

Given a set of activities T, an event log over T is a multiset $L : T^* \to \mathbb{N}$. A sequence $\sigma \in T^*$ is a called *trace*. A trace σ is contained in a log if $L(\sigma) \geq 1$. Given a trace $\sigma = t_1, t_2, \ldots, t_n$, and a natural number $1 \leq k \leq n$, the sequence $t_1, t_2, \ldots, t_k$ is called the *prefix* of length k in σ. Given a log L, we denote by *Pref*(L) the set of all prefixes of traces in L. Finally, $\#(\sigma, e)$ is the number of times that activity e occurs in sequence σ.

2.1 Abstract Interpretation

Intuitively, abstract interpretation defines a procedure to compute an upper approximation for a given behavior of a system that still suffices for reasoning about the behavior itself. An important decision is the choice of the kind of upper approximation to be used, which is called the *abstract domain.* For a given problem, there are typically several abstract domains available. Each abstract domain provides a different trade-off between precision (closeness to the exact result) and computational efficiency.

There are many problems where abstract interpretation can be applied, several of them oriented towards the compile-time detection of run-time errors in software. For example, some analysis based on abstract interpretation can discover numeric invariants among the variables of a program. Several abstract domains can be used to describe the invariants: intervals [9], octagons [14], convex polyhedra [11], among others. These abstract domains provide different ways to approximate sets of values of numeric variables. For example, Figure 2 shows how these abstract domains can represent the set of values of a pair of variables x and y. For space reasons, we focus on the abstract domain of convex polyhedra. In the experiments, the domain of octagons is also used.

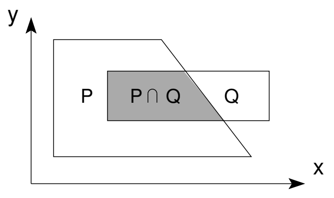


Interval	**Octagon**	**Convex polyhedron**
$0 \leq x \leq 2$	$0 \leq x \leq 2$	$y - x \geq 0$
$0 \leq y \leq 4$	$0 \leq y - x \leq 2$	$x - 2y \geq 0$
		$3x - y \leq 2$

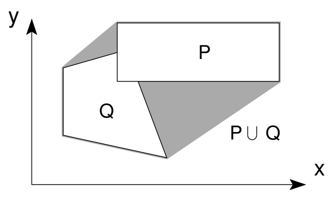
Fig. 2. Approximating a set of values (left) with several abstract domains

Convex Polyhedra. This domain can be described as the sets of solutions of a set of *linear inequality constraints* with rational ($\mathbb{Q}$) coefficients. Let P be a polyhedron over $\mathbb{Q}^n$; then it can be represented as the solution to some system of m inequalities $P = \{X | AX \leq B\}$ where $A \in \mathbb{Q}^{m \times n}$ and $B \in \mathbb{Q}^m$.

The domain of convex polyhedra provides the operations required in abstract interpretation. In this paper, we will mainly use the following two operations:



Meet ($\cap$): Given convex polyhedra P and Q, their meet is the intersection $P \cap Q$. Notice that this operation is exact, e.g., the meet or intersection of two convex polyhedra is always a convex polyhedron.



Join ($\cup$): Given convex polyhedra P and Q, we would like to compute the union of P and Q. Unfortunately the union of convex polyhedra is not necessarily a convex polyhedron. Therefore, the union of two convex polyhedra is approximated by the *convex hull*, the smallest convex polyhedron that includes both operands, denoted by $P \cup Q$. The example on the left shows $P \cup Q$ in gray.

2.2 Estimation, Drift, and Change Detection

In a stream setting one receives a potentially infinite stream of objects X_1, X_2, $\ldots X_t \ldots$ to be analyzed, where object X_t becomes available only at time step t. The underlying assumption is that there is some distribution D on the set of all objects generating the X_i's, often together with the assumption that some degree of independence among the draws exist. The *concept drift* problem occurs when the distribution D cannot be assumed to be stationary, but there is in fact a distribution D_t for every t, which change gradually or abruptly over time.

There are two common strategies for dealing with concept drift: in the *sliding window* approach, one keeps a window of the last W elements, and the *change detection* approach, where one *estimates* or monitors some statistics of the X_i's and when some large enough deviation from past behavior is detected, change is declared. These two strategies can, of course, be combined too.

In this paper we will use for estimation and change detection the ADWIN (ADaptive WINdowing) method proposed in [1]. This choice is not the essential to the paper, and other methods (such as CUSUM or Page-Hinkley). Roughly speaking, ADWIN reads a real number X_t at each time step t, outputs an estimation of the current average of $E[X_t]$ in D_t, and also outputs a bit indicating whether drift has been detected in the recent observations. Internally, it keeps a window of the most recent X_t's whose length varies adaptively. It requires no parameters such as window length, delivering the user from the difficult tradeoff in such choices, and is efficient in the sense that it simulates a window of length W using $O(\log W)$ memory (rather than the obvious $O(W)$) and amortized $O(1)$ time per item. Furthermore, unlike most methods which are heuristics, ADWIN has rigorous guarantees (theorems) on its change detection performance. See [1] and the extended version of this paper for details[2].

3 Concept Drift Detection via Abstract Interpretation

This section describes the technique for concept drift detection in process mining. It first learns a process model using abstract interpretation (Section 3.1), which will be the main actor for the concept drift detection method in Section 3.2.

3.1 Learning via Abstract Interpretation

We now introduce the element to link traces from a log and abstract interpretation, which was initially presented in [7]:

Definition 1 (Parikh vector). *Given a trace $\sigma \in \{t_1, t_2, \dots, t_n\}^*$, the* Parikh vector *of σ is defined as $\widehat{\sigma} = (\#(\sigma, t_1), \#(\sigma, t_2), \dots, \#(\sigma, t_n))$.*

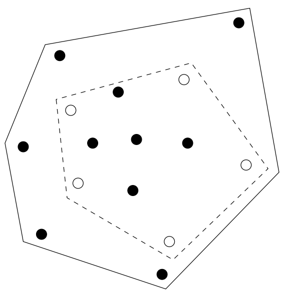
Any component of a Parikh vector can be seen as a constraint for the n-dimensional point that it defines. Hence, the Parikh vector defined by $\widehat{\sigma} = (\#(\sigma, t_1), \#(\sigma, t_2), \dots, \#(\sigma, t_n))$, a point, can be seen as the polyhedron $P_{\widehat{\sigma}} = \bigcap_{i=1}^{n}(x_i = \#(\sigma, t_i))$, where each variable x_i denotes the number of occurrences of activity t_i in σ, i.e., $x_i = \#(\sigma, t_i)$[3]. For each prefix σ of a trace in L, a polyhedron $P_{\widehat{\sigma}}$ can be obtained. Given all possible prefixes $\sigma_1, \sigma_2, \dots, \sigma_m$ of traces

[2] Extended version of the paper:
`http://www.lsi.upc.edu/~jcarmona/ida2012ext.pdf`

[3] Hence a point $\widehat{\sigma}$ is represented as the polyhedron $P_{\widehat{\sigma}}$ that defines it.

in L, the polyhedra $P_{\widehat{\sigma_1}}, P_{\widehat{\sigma_2}}, \ldots, P_{\widehat{\sigma_k}}$ can be found[4]. Finally, the polyhedron $P = \bigcup_{i \in \{1 \ldots m\}} P_{\widehat{\sigma_i}}$ can be learned as the *convex-hull* of the points represented by the polyhedra $P_{\widehat{\sigma_1}}, P_{\widehat{\sigma_2}}, \ldots, P_{\widehat{\sigma_m}}$, and taken as the representation of the log.

Computing the convex hull of a large set of points may be expensive, so subsampling has been suggested [7] as a way to speed-up the process. The figure on the right shows an example: when the polyhedra of all the points are united, it may derive a polyhedron that covers large areas void of real points. If instead, a random sample of five points is used (denoted by points with white background), a smaller polyhedron is derived which, although not including the complete set of Parikh vectors from the log, represents a significant part of it and therefore the percentage of areas void of real points may be reduced. The *mass* of the polyhedron is the probability that a Parikh vector from the log belongs to the polyhedron; obviously, mass close to 1 is desired.



3.2 Online Algorithm for Concept Drift Detection

The technique is described as Algorithm 1. The input of the algorithm is the sequence of points or Parikh vectors produced from the traces received. The algorithm is divided into three stages: Learn (lines 2-8), Mean estimation (lines 9-15) and Mean monitoring (lines 16-25). Notice that the algorithm iterates over these three stages each time a drift is detected (Line 23).

In the learning stage, a set of m points is collected for training. The larger the m, the less biased the sampling is to a particular behavior, since the distribution of points will be more diverse. The outcome of the learning stage is a polyhedron $\widehat{P}$ that represents the concept underlying the set of points from the input.

In the mean estimation stage, the mass (fraction of points) belonging to $\widehat{P}$ is estimated using an ADWIN instance W. This process is iterated until a convergence criteria is met, e.g., the mean is stabilized to a given value. Notice that in this stage the algorithm, as written, may not converge to a stationary value if change keeps occurring. One could prevent this problem by using e.g. Chernoff bounds to bound on the number of iterations to reach a given approximation, assuming stationariness. In the monitoring stage, the same estimation is continued, but now with the possibility to detect a drift by detecting a change in the mass of $\widehat{P}$. In the simplest version, when a drift is detected, the three-stage technique is re-started from scratch; better strategies will be discussed in Section 5.

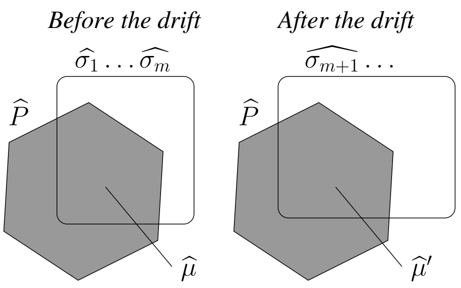
As explained in Section 2.2, an ADWIN instance W can be used to monitor a data sequence in order to determine drifts. In our setting, the data sequence will be produced by the outcome of the following question performed on each point from the traces in the log: does the learned polyhedron include the point? If it does, we will update W with a 1, and otherwise with a 0.

[4] Here k is in practice significantly smaller than $\sum_{\sigma \in L} |\sigma|$ since many prefixes of different traces in L share the same Parikh vector.

Algorithm 1. Concept Drift Detection algorithm

```
Input: Sequence of Parikh vectors $\widehat{\sigma_1}, \widehat{\sigma_2}, \ldots$
begin
    Select appropriate training size $n$;
    Collect a random sample of $m$ vectors out of the first $n$;
    // $m < n$ if required for efficiency
    // Stage 1: Learning the current concept $\widehat{P}$
    $\widehat{P}$ = "empty domain" ;
    for $j \leftarrow 1$ to $m$ do
        let $\widehat{\sigma_j}$ be the $j$th randomly selected vector; compute $P_{\widehat{\sigma_j}}$;
        $\widehat{P} = \widehat{P} \cup P_{\widehat{\sigma_j}}$;
    end
    // Stage 2: Estimating the average of points included in $\widehat{P}$
    $W$ =InitADWIN;
    $i = m + 1$;
    repeat
        if $\widehat{\sigma_i} \in \widehat{P}$ then $W = W \cup \{1\}$;
        else $W = W \cup \{0\}$;
        $i = i + 1$;
    until convergence criteria on $W$ estimation;
    // Stage 3: Monitoring the average of points included in $\widehat{P}$
    while true do
        if $\widehat{\sigma_i} \in \widehat{P}$ then $W = W \cup \{1\}$;
        else $W = W \cup \{0\}$;
        $i = i + 1$;
        if Drift detected on $W$ then
            Declare "drift!";
            Throw away the current set of points;
            Jump to line 2;
        end
    end
end
```

This way, W will estimate the mass of current polyhedron, denoted $\widehat{\mu}$. The figure next illustrates the approach: a polyhedron $\widehat{P}$ which (maybe partially) represents the Parikh vectors of the traces in the log is learned, and we estimate $\widehat{\mu}$, the fraction of the points falling into the polygon. When a change occurs (in the figure, more points are added), chances are that this quantity changes to a new value $\widehat{\mu'}$ (this assumption will be discussed in Section 5).



3.3 Rigorous Guarantees

Using the rigorous guarantees of ADWIN given in [1], one can give a rigorous statement on the ability of the method above to detect drift in its input. The proof can be found in the extended version.

Theorem 1. *Suppose that for a sufficiently long time* $T_0 \dots T_1$ *the input distribution has remained stable, and that the learn phase has built a polyhedron* $\widehat{P}$*, with mass* μ_1*. Suppose that by time* T_2 $(> T_1)$ *the input distribution has changed so that the mass of* $\widehat{P}$ *is* μ_2 *from then on. Then by time at most* $T_2 + O(\ln(T_2 - T_0)/(\mu_2 - \mu_1)^2)$ *the method will have detected change and restarted.*

Note that the case of abrupt change is when $T_2 = T_1 + 1$, and that larger changes in the mass of the current polyhedron imply shorter reaction times.

4 Experiments on Concept Drift Detection

To test the detection technique of Section 3, a set of models (in our case, Petri nets) have been used. For each model M, a log has been created by simulating M. The first six models in Table 1 are taken from [8], and represent typical behaviors in a concurrent system. Model Cycles(X,Y) represents a workflow of overlayed cyclic processes. Finally, the models a12f0n00 ... t32f0n00 are originated from well-known benchmarks in the area of process mining.

To derive logs that contain a drift, each model has been slightly modified in four different dimensions:

- *Flip*: the ordering of two events of the model has been reversed.
- *Rem*: one event of the model has been removed
- *Conc*: two sequential events have been put in parallel
- *Conf*: two sequential or concurrent events have been put in conflict

These transformations represent a wide spectra of drifts for control-flow models. Each transformation leads to a new model, which can be simulated to derive the corresponding log[5]. An assumption of this work, which is inherited from the use of the techniques in [7], is that a drift causes a modification in the spectra of Parikh vectors representing a model. Although this assumption is very likely in most practical cases, it does not cover the drifts which neither incorporate nor remove Parikh vectors to the current spectra. However, using an alternative technique to [7] in Algorithm 1 which is sensitive to this type of drifts can be the cure for this particular problem. Notice that only a single transformation is applied to a model to introduce drift. In general, however, drift may arise from

[5] All the models and logs used in this paper can be obtained at the following url: `http://www.lsi.upc.edu/~jcarmona/benchsIDA2012.tar`

Table 1. Concept drift detection: number of points to detect the drift

benchmark	$\|\Sigma\|$	$\|Places\|$	$\|L1\|$	*Flip*	*Rem*	*Conc*	*Conf*
SharedRes(6)	24	25	4000	115	54	183	37
SharedRes(8)	32	33	4000	165	73	381	83
ProdCons(8)	41	42	4000	337	550	262	266
ProdCons(9)	46	47	4000	256	136	323	489
WeightMG(9)	9	16	4000	101	16	75	16
WeightMG(10)	10	18	4000	147	28	53	18
Cycles(4,2)	14	11	4000	563	23	664	22
Cycles(5,2)	20	16	4000	554	22	845	21
a12f0n00	12	11	620	83	76	117	15
a22f0n00	22	19	2132	340	56	99	198
a32f0n00	32	32	2483	67	79	258	162
a42f0n00	42	46	3308	178	41	185	37
t32f0n00	33	31	3766	143	28	394	36

several transformations, either simultaneous or spaced out in time. However, if the technique is able to identify drift from a single transformation, it should also be able to detect these more drastic drifts.

The experiment performed is to use the concatenation of two logs $L1$, $L2$ with different distribution, so that abrupt change occurs at the transition from $L1$ to $L2$. Using the Apron library [13], an octagon/polyhedra P is obtained from $L1$ by sampling a few points, and an ADWIN is then created to estimate the fraction of points in the input (still from $L1$) that are covered by P. When the estimation converges, the input is switched to points from $L2$. If the drift is not sporadic, the fraction of points covered by P will change, ADWIN will detect change in this quantity, and drift will be declared.

In Table 1 the results of the experiment are provided. The second column in the table reports the number of different events (which also represents the number of transitions in the Petri net), and the third column reports the number of places. Column $|L1|$ provides the number of points used in the stages 1 and 2 of the algorithm (we have set a maximum of four thousand points in the simulation)[6]. The following columns denote the logs corresponding to the models with each one of the aforementioned transformations. In each cell from column five on we provide the number of points needed to sample in order to detect a drift. For instance, for the SharedRes(8) benchmark, it only needs to sample 73 points in order to detect a drift when a single event is removed, and needs to sample 381 points to detect that two events became concurrent. In terms of CPU time, all drifts have been detected in few seconds[7].

[6] $|L2|$ is particular to each drift. In general it is of the same magnitude as $|L1|$.
[7] In the experiments the domain of octagons has been used when $|\Sigma| > 20$, to bound the time and memory requirements in the learning stage.

A second experiment was performed on a log with another kind of drift. The detailed description can be found in [6]. It is a real log containing the processing of copy/scan jobs on a digital copier. The log contains 1050 traces and 35 event classes, with a drift introduced after 750 traces, consisting in the addition of a new functionality to the copier (zooming function for image processing). For the first part of the log (traces $1 - 750$), there are 34.427 points, whereas for the second part (traces $751 - 1050$), 13.635 points. The technique of this paper identifies the drift by sampling only 504 points.

5 Change Location and Unraveling Process Evolution

So far, we have focused into explaining how to apply the theory of abstract interpretation together with estimation and change detection in order to detect concept drifts in the area of process mining. In this section we will briefly address the two other problems highlighted in [5]: change location and characterization, and unravel process evolution; this is ongoing work.

Change Location and Characterization. A polyhedron P is the solution to the system of m inequalities $P = \{X|AX \leq B\}$ where $A \in \mathbb{Q}^{m \times n}$ and $B \in \mathbb{Q}^m$ (see Section 2.1). A subset C of these inequalities called *causal constraints* can be used to derive the corresponding process model (see [7] for details). A causal constraint satisfies particular conditions that makes it possible to be converted into a process model element, e.g., a *place* and its corresponding arcs in a Petri net. Since the adaptive windowing technique described in Section 2.2 requires few resources, one can use one adaptive window *for monitoring each causal constraint in C*. Thus, after the learning stage of Algorithm 1, $|C|$ instances of ADWIN are used to estimate the average satisfaction for each constraint. Finally, in stage three these ADWIN's can detect drift at each of the constraints. When global or partial drift occurs, its location is exactly characterized by the causal constraints that have experienced drift, which can be mapped to the corresponding places in the process model. Remarkably, this provides a *fine-grain* concept drift detection version of the technique presented in Section 3.

Unraveling Process Evolution. After change has been localized and characterized as above, the new process model can be then produced. This is crucial to unravel the process evolution [5]. Two sets of causal constraints will be used to derive the new process model: i) the causal constraints which still are valid after the drift, and ii) the new set of causal constraints that may appear in the new polyhedron learned by revisiting stage one of Algorithm 1. For the former set, both drifting and non-drifting causal constraints detected in the previous iteration of Algorithm 1 will be considered. For drifting causal constraints, a threshold value may be defined to determine when drift is strong enough to invalidate it. As for the complexity of the model revision, for example the method in [7] for deriving Petri nets from polyhedra is well-behaved in the sense that a change in some of the inequalities can be translated to a local change in the Petri net, with proportional computational cost.

The same idea can be used to alleviate a problem with the change detection strategy described in Section 3. Recall that there we were only detecting changes where new points appeared in regions outside the learned polyhedron. Drift may also mean that points previously in the log, or in its convex hull, do no longer appear (e.g., if some behaviors disappear and the polygon becomes larger than necessary). We can detect many such changes by monitoring many aspects of the stream of points, instead of just the mass of the learned polyhedron. For example, we could use an array of ADWIN instances to monitor the average distance among points, distance to their centroid or set of designated points, distance to each constraint, projection to a set of random hyperplanes, etc.

6 Conclusions and Future Work

Concept drift is an important concern for any data analysis scenario involving temporally ordered data. Surprisingly, there is very little work (in fact, possibly only [5]) in dealing with concept drift within process mining techniques.

In this paper we have presented the first online mechanism for detecting and managing concept drift, combined with the process mining approach in [7] based on abstract interpretation and Petri net models. Our experiments on process mining benchmark data twisted to incorporate drift show that our method detects abrupt changes quickly and accurately. We have also described how to apply the mechanism for a richer set of tasks: characterizing and locating change, unraveling the process change, and revising the mined models.

Future work includes experimenting with different forms of change, particularly, gradual, long term changes and those discussed at the end of Section 5; implementing the fine-grain detection mechanisms for change location and unraveling; and using our approach in a real scenario with a high volume of data, so sampling becomes essential, and with strong requirements on time and memory. Also, investigating tailored techniques that can deal with logs that contain *noise* is an interesting future research direction. A possibility will be to adapt Algorithm 1 to use some of the few process discovery techniques that can handle noise in the log [12, 19, 20].

Acknowledgements. We would like to thank R. P. Jagadeesh Chandra Bose and W. M. P. van der Aalst for the providing some of the logs of the experiments. This work is partially supported by BASMATI MICINN project (TIN2011-27479-C04-03), by the SGR2009-1428 (LARCA), by the EU PASCAL2 Network of Excellence (FP7-ICT-216886), by the FORMALISM project (TIN2007-66523) and by the project TIN2011-22484.

References

1. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: SDM. SIAM (2007)
2. Bifet, A., Gavaldà, R.: Adaptive Learning from Evolving Data Streams. In: Adams, N.M., Robardet, C., Siebes, A., Boulicaut, J.-F. (eds.) IDA 2009. LNCS, vol. 5772, pp. 249–260. Springer, Heidelberg (2009)
3. Bifet, A., Gavaldà, R.: Mining frequent closed trees in evolving data streams. Intell. Data Anal. 15(1), 29–48 (2011)
4. Bifet, A., Holmes, G., Pfahringer, B., Gavaldà, R.: Mining frequent closed graphs on evolving data streams. In: Apté, C., Ghosh, J., Smyth, P. (eds.) KDD, pp. 591–599. ACM (2011)
5. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P., Žliobaitė, I.e., Pechenizkiy, M.: Handling Concept Drift in Process Mining. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 391–405. Springer, Heidelberg (2011)
6. Jagadeesh Chandra Bose, R.P.: Process Mining in the Large: Preprocessing, Discovery, and Diagnostics. PhD thesis, Eindhoven University of Technology (2012)
7. Carmona, J., Cortadella, J.: Process Mining Meets Abstract Interpretation. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part I. LNCS, vol. 6321, pp. 184–199. Springer, Heidelberg (2010)
8. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Trans. on Computers 59(3), 371–384 (2009)
9. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: 2nd Int. Symposium on Programming, Paris, France, pp. 106–130 (1976)
10. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, pp. 238–252. ACM Press (1977)
11. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proc. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, pp. 84–97. ACM Press, New York (1978)
12. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
13. Jeannet, B., Miné, A.: Apron: A Library of Numerical Abstract Domains for Static Analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
14. Miné, A.: The octagon abstract domain. In: IEEE Analysis, Slicing and Tranformation, pp. 310–319. IEEE CS Press (October 2001)
15. Murata, T.: Petri nets: Properties, analysis and applications. Proc. of the IEEE 77(4) (1989)
16. van der Aalst, W., et al.: Process Mining Manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012)

17. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
18. van der Aalst, W.M.P., Günther, C.W.: Finding structure in unstructured processes: The case for process mining. In: Basten, T., Juhás, G., Shukla, S.K. (eds.) ACSD, pp. 3–12. IEEE Computer Society (2007)
19. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.T.: Genetic Process Mining. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 48–69. Springer, Heidelberg (2005)
20. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: CIDM, pp. 310–317 (2011)