Dynamic Constructs Competition Miner - Occurrence- vs. Time-Based Ageing

David Redlich[1,2(✉)], Thomas Molka[1,3], Wasif Gilani[1], Gordon Blair[2], and Awais Rashid[2]

[1] SAP Research Center Belfast, Belfast, UK
{david.redlich,thomas.molka,wasif.gilani}@sap.com
[2] Lancaster University, Lancaster, UK
{gordon,marash}@comp.lancs.ac.uk
[3] University of Manchester, Manchester, UK

Abstract. Since the environment for businesses is becoming more competitive by the day, business organizations have to be more adaptive to environmental changes and are constantly in a process of optimization. Fundamental parts of these organizations are their business processes. Discovering and understanding the actual execution flow of the processes deployed in organizations is an important enabler for the management, analysis, and optimization of both, the processes and the business. This has become increasingly difficult since business processes are now often dynamically changing and may produce hundreds of events per second. The basis for this paper is the Constructs Competition Miner (CCM): A divide-and-conquer algorithm which discovers block-structured processes from event logs possibly consisting of exceptional behaviour. In this paper we propose a set of modifications for the CCM to enable dynamic business process discovery of a run-time process model from a stream of events. We describe the different modifications with a particular focus on the influence of individual events, i.e. ageing techniques. We furthermore investigate the behaviour and performance of the algorithm and the ageing techniques on event streams of dynamically changing processes.

Keywords: Run-time models · Business Process Management · Process mining · Complex Event Processing · Event streaming · Big Data

1 Introduction

The success of modern organizations has become increasingly dependent on the efficiency and performance of their employed business processes (BPs). These processes dictate the execution order of singular tasks to achieve certain business goals and hence represent fundamental parts of most organizations. In the context of business process management, the recent emergence of Big Data yields new challenges, e.g. more analytical possibilities but also additional run-time constraints. An important discipline in this area is Process Discovery: It is concerned

P. Ceravolo et al. (Eds.): SIMPDA 2014, LNBIP 237, pp. 79–106, 2015.
DOI: 10.1007/978-3-319-27243-6_4

with deriving process-related information from event logs and, thus, enabling the business analyst to extract and understand the actual behaviour of a business process. Even though they are now increasingly used in commercial settings, many of the developed process discovery algorithms were designed to work in a static fashion, e.g. as provided by the ProM framework [21], but are not easily applicable for processing real-time event streams. Additionally, the emergence of Big Data results in a new set of challenges for process discovery on event streams, for instance [16,22]: (1) *high event frequency* (e.g. thousands of events per second), and (2) *less rigid processes* (e.g. BPs found on the operational level of e-Health and security use-cases are usually subjected to frequent changes).

In order to address the challenges we propose modifications for the Constructs Competition Miner (CCM) [15] to enable dynamic process discovery as proposed in [16]. The CCM is a process discovery algorithm that follows a divide-and-conquer approach to directly mine a block-structured process model which consists of common BP-domain constructs and represents the main behaviour of the process. This is achieved by calculating global relations between activities and letting different *constructs* compete with each other for the most suitable solution from top to bottom using "soft" constraints and behaviour approximations. The CCM was designed to deal with noise and not-supported behaviour. To apply the CCM on event streams the algorithm was split up into two individually operating parts:

1. **Run-Time Footprint Calculation**, i.e. the current footprint[1], which represents the abstract "state" of the system, is updated with occurrence of each event. The influence of individual events on the run-time footprint is determined by two different strategies: time-based and occurrence-based ageing. Since every occurring event constitutes a system state transition, the algorithmic execution-time needs to be kept to a minimum.
2. **Scheduled Footprint Interpretation**, i.e. from the footprint the current business process is discovered in a scheduled, reoccurring fashion. Since this part is executed in a different lifecycle it has less execution-time constraints. In this step the abstract "computer-centric" footprint is transformed into a "human-centric" business process representation.

The remainder of this paper provides essential background information (Sect. 2), a discussion of related work (Sect. 3), a summarized description of the original CCM (Sect. 4), the modifications that were carried out on top of the CCM to enable Scalable Dynamic Process Discovery (Sect. 5), an evaluation of the behaviour of the resulting algorithm for event streams of dynamically changing processes (Sect. 6), and an outlook of future work (Sect. 7).

2 Background

Business Processes are an integral part of modern organizations, describing the set of activities that need to be performed, their order of execution, and the

[1] footprint is a term used in the process discovery domain, abstractly representing existent "behaviour" of a log, e.g. activity "a" is followed by activity "b".

entities that execute them. Prominent BP examples are Order-to-Cash or Procure-to-Pay. According to Ko et al. BPs are defined as *"...a series or network of value-added activities, performed by their relevant roles or collaborators, to purposefully achieve the common business goal"* [8]. A BP is usually described by a *process model* conforming to a business process standard, e.g. Business Process Model and Notation (BPMN) [14], or Yet Another Workflow Language (YAWL) [19]. In this paper, we will focus on business processes consisting of a set of common control-flow elements, supported by most of the existing BP standards: start and end events, activities (i.e. process steps), parallel gateways (AND-Split/Join), and exclusive gateways (XOR-Split/Join) (see [14,19]). In Fig. 1 an example process involving all the introduced elements is displayed. Formally, we define a business process model as follows [15]:

Definition 1. *A business process model is a tupel* $BP = (A, S, J, E_s, E_e, C)$ *where* A *is a finite set of activities,* S *a finite set of splits,* J *a finite set of joins,* E_s *a finite set of start events,* E_e *a finite set of end events, and* $C \subseteq F \times F$ *the path connection relation, with* $F = A \cup S \cup J \cup E_s \cup E_e$*, such that*

- $C = \{(c_1, c_2) \in F \times F \mid c_1 \neq c_2 \wedge c_1 \notin E_e \wedge c_2 \notin E_s\}$,
- $\forall a \in A \cup J \cup E_s : |\{(a, b) \in C \mid b \in F\}| = 1$,
- $\forall a \in A \cup S \cup E_e : |\{(b, a) \in C \mid b \in F\}| = 1$,
- $\forall a \in J : |\{(b, a) \in C \mid b \in F\}| \geq 2$,
- $\forall a \in S : |\{(a, b) \in C \mid b \in F\}| \geq 2$, *and*
- *all elements* $e \in F$ *in the graph* (F, C) *are on a path from a start event* $a \in E_s$ *to an end event* $b \in E_e$.

For a block-structured BP model it is furthermore required that the process is hierarchically organised [15], i.e. it consists of unique join-split-pairs, each representing either a single entry or a single exit point of a non-sequential BP construct, e.g. Choice, Parallel, Loop, etc. The example process in Fig. 1 is a block-structured process. A similar representation gaining popularity in recent years is the *process tree*, as defined based on Petri nets/workflow nets in [9].

When a business process is automatically or semi-automatically executed with a BP execution engine, e.g. with a Business Process Management System (BPMS), an *event log* is produced, i.e. a all occurred events are logged and

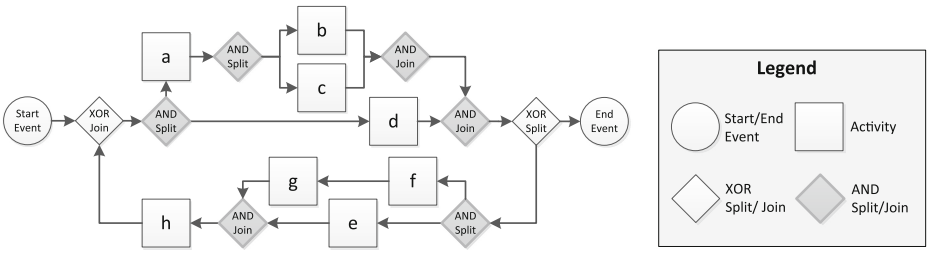

Fig. 1. Example business process with all element types included

stored. These logs and their contained events may capture different aspects of a process execution, e.g. a different granularity of events are logged. In this paper however, we only focus on a minimal set of event features: In order to allow the discovery of the control-flow, every event is required to have a reference (1) to the associated *process instance* and (2) to the corresponding *activity*. Furthermore, we assume that the log contains exactly one event for each activity execution, i.e. activity lifecycle events are not regarded. All events resulting from the execution of the same *process instance* are captured in one *trace*. A trace is assumed to be independent from other traces, i.e. the execution order of a process instance is not in any way dependent on the execution of a second instance. Accordingly, an event e is represented by a pair $e = (t, a)$ where $t \in \mathbf{N}$ is the unique identifier of the trace and $a \in A$ is a unique reference to the executed activity.

The research area of *Process Discovery* is concerned with the extraction of a business process model from event logs without using any a-priori information [23]. Conventional challenges in process discovery originate from the motivation to achieve a high quality of results, i.e. discovered processes should support as accurately as possible the behaviour contained in the log. In particular that means, process discovery algorithms have to deal with multiple objectives, e.g. precision, simplicity, fitness - over-fitting vs. under-fitting (see [23]). Process discovery algorithms are usually assumed to be carried out in an static way as an "offline" method. This is reflected by the fact that the input for these algorithms is an entire log as conceptually shown by the following definition:

Definition 2. *Let the log* $L_n = [e_0, e_1, ...e_n]$ *be a sequence of* $n+1$ *events ordered by time of occurrence* $(\forall i < j \wedge e_i, e_j \in L_n : time(e_i) \leq time(e_j))$ *and* BP_n *be the business process model representing the behaviour in* L_n*, then process discovery is defined as a function that maps a log* L_n *to a process* BP_n*:*

$$ProcessDiscovery : [e_0, e_1, ..., e_n] \Rightarrow BP_n$$

3 Related Work

A large number of process discovery algorithms exist, e.g. Inductive Miner [9], HeuristicsMiner [25], alpha-miner [20] and CCM [15]. These and many algorithms have in common that at first a *footprint* of the log is created based on which the process is constructed. Similar to the CCM, the following related algorithms also discover block-structured processes: (1) Genetic process discovery algorithms that restrict the search space to block-structured process models, e.g. [4]. However, these are non-deterministic and generally have a high execution time due to exponentially expanding search space. (2) Another relevant approach that is conceptually similar to the CCM is proposed in [9], the Inductive Miner (IM): A top-down approach is applied to discover block-structured Petri nets. The original algorithm evaluates constraints based on local relationships between activities in order to identify the representing construct in an inductive fashion. In recent work, the IM has also been extended to deal with

noise [10]. Generally, in all discovery approaches based on footprints known to the authors the footprint is represented by a *direct neighbours* matrix representing information about the local relations between the activities, e.g. for the BP of Fig. 1: h can only appear *directly* after g or e. As discussed in Sect. 4 the CCM on the other hand extracts the process from a footprint based on global relations between activities, e.g. h appears *at some point* after g or e.

However, of little importance for conventional process discovery algorithms is their practicality with regards to an application during run-time: as defined in Definition 2 process discovery is a static method that analyses an event log in its entirety. An alternative to this approach is the immediate processing of events when they occur to information of an higher abstraction level in order to enable a real-time analysis. This approach is called Complex Event Processing (CEP): a method that deals with the event-driven behaviour of large, distributed enterprise systems [11]. More specifically, in CEP events produced by the systems are captured, filtered, aggregated, and finally abstracted to complex events representing high-level information about the situational status of the system, e.g. performance, control-flow, etc. The need for monitoring aspects of business processes at run-time by applying CEP methodologies has been identified by Ammon et al., thus coining the term Event-Driven Business Process Management (EDBPM) - a combination of two disciplines: Business Process Management (BPM) and Complex Event Processing [1]. The dynamic process discovery solution proposed in this paper is an application of EDBPM (see Sect. 5). The motivation is to have a run-time reflection of the employed processes based on up-to-date rather than historical information which essentially allows business analysts to react quicker to changes or occurring bottlenecks etc. in order to optimise the overall performance of the monitored processes. In accordance to this objective process discovery algorithms for event streams have to deal with two additional challenges as opposed to the traditional process discovery algorithms:

1. The application of process discovery on event streams is executed in a real-time setting and thus is required to conform to special memory and execution time constraints. Especially with regards to many modern systems producing "big data", i.e. data that is too large and complex to store and process all of it [12]. This means in particular, that online algorithms should be able to (1) process an infinite number of events without exceeding a certain memory threshold and (2) process each event within a small and near-constant amount of time [5].
2. Real-life BPs are often subject to externally or internally initiated change which has to be reflected in the results of an process discovery algorithm analysing an event stream. The observed characteristic of dynamically changing processes is also called *concept drift* and has been identified as a major challenge for process discovery algorithms [2,22,23]. Generally, discovery algorithms on event streams should be able to (1) reflect newly appearing behaviour as well as (2) forget outdated behaviour.

Although not specifically, *incremental process mining* as introduced in [3] does attempt to anticipate the problem of concept drift to some extent. Here,

the assumption is that a log does not yet contain the entire behaviour of the process (i.e. an incomplete log) at the time of the discovery of the initial (declarative) process. Additional behaviour, that occurred after the initial discovery and captured in a second log, is analysed separately and the new information is then added to the existing BP model. This is possible due to structure of declarative BP specifications. The process of incrementally analysing log segments and then extending the BP model accordingly, i.e. incremental process mining, is motivated by the assumption that the update of an already existing (declarative) BP model is easier than to always analyse the complete log from scratch [3]. Another approach called *incremental worklfow mining* is based on the same principle but does discover and adapt a Petri Net from incrementally processing log segments [6,7]. It is a semi-automatic (and prototypical) approach specifically designed for dealing with process flexibility in *Document Management Systems* that does not anticipate incomplete or noisy logs. A third incremental approach is presented in [18] which utilises the theory of regions to create transition systems for successive sub-logs and eventually transform them into a Petri Net. Albeit based on a slightly different concept, incremental process mining approaches can be considered for process discovery on event streams since the event processing could be designed to group a number of successive traces into sub-logs which are then individually analysed and incrementally extend the overall BP. However, a conceptual weakness of incremental mining approaches is the lacking ability of forgetting outdated behaviour.

In the context of process discovery on event streams, a synonymous term sometimes used is *Streaming Process Discovery* (SPD). SPD was coined by Burattin et al. in [5]. In their work the HeuristicsMiner [25] has been modified for this purpose and a comprehensive evaluation of different event stream processing types was carried out. The fundamentals of the HeuristicsMiner remain the same but the direct-neighbours-footprint is dynamically adapted or rebuilt while processing the individual events. From this the Causal Net is periodically extracted, e.g. for every event or every 1000 events, using the traditional HeuricticsMiner. For instance, for the evaluation of the different streaming methods the HM discovery was triggered every 50 events [5]. Three different groups of event streaming methods have been implemented and investigated:

Event Queue: The basic methodology of this approach is to collect events in a queue which is representing a log that can be analysed in the traditional way of process discovery. Three basic types can be differentiated: (1) In the *sliding window* approach the queue is a FIFO (First-In-First-Out), i.e. when the maximum queue length (queue memory) is reached for every new event inserted, the oldest event in the queue is removed; (2) In the *periodic reset* approach the queue is reset whenever the maximum queue length is reached. The main advantage of these approaches are that the event queue can be regarded as event log and enables the analysis via traditional discovery/mining algorithms on event logs. Two of the main disadvantages are: Each event is handled at least twice: once to store it in the queue and once or more to discover the model from the queue; Also, it does only allow for a rather simplistic interpretation of “history”,

i.e. an older event is either still in the queue and has same influence as a newer event, or it is completely forgotten.

Stream-Specific Approaches: Stream-specific approaches already process events into footprint information, i.e. queues that consist of a fixed size hold information about the latest occurring activities and directly-follows relations. When a new event occurs all values in the queues are updated and/or replaced. Burattin et al. distinguish between the following three update operations: (1) *Stationary*, i.e. the queues function as a "sliding window" over the event stream and every queue entry has the same weight, (2) *Ageing*, i.e. the weight of the latest entry is increased and the weights of older entries in the queue are decreased, and (3) *Self-Adaptive Ageing*, i.e. the factor with which the influence of older entries decreases is dependent on the fitness of the discovered model in relation to latest events stored in an additional sample queue of a fixed size: quickly decreasing for a low fitness and slowly decreasing for a high fitness. Generally, stream-specific approaches are assumed to be a more balanced approach since events are only handled once and directly processed into footprint information [5]. Burattin et al. also argue that ageing-based approaches have a more realistic interpretation of "history" since older events have less influence than newer events [5]. One disadvantage is that the footprint is captured through a set of queues with a fixed size: if this size is set too low, behaviour is prematurely forgotten; if this size is set too high some of the old behaviour is never forgotten.

Lossy Counting: Lossy Counting is a technique adopted and modified from [13] that uses approximate frequency count and divides the stream into a fixed number of buckets.

Another approach for discovering concept drifts on event streams of less relevance to the paper's topic is presented in [12]: A discovery approach for declarative process models using the sliding window approach and lossy counting to update a set of valid business constraints according to the events occurring in the stream.

4 Static Constructs Competition Miner

The CCM as described in [15] is a deterministic process discovery algorithm that operates in a static fashion and follows a divide-and-conquer approach which, from a given event log, directly mines a block-structured process model that represents the main behaviour of the process. The CCM has the following main features [15]: (1) A deadlock-free, block-structured business process without duplicated activities is mined; (2) The following BP constructs are supported and can be discovered for single activities: Normal, Optional, Loopover, and Loopback; or for a set of activities: Choice, Sequence, Parallel, Loop, Loopover-Sequence, Loopover-Choice, Loopover-Parallel (see Fig. 2), and additionally all of them as optional constructs - these are constructs supported by the majority of business process

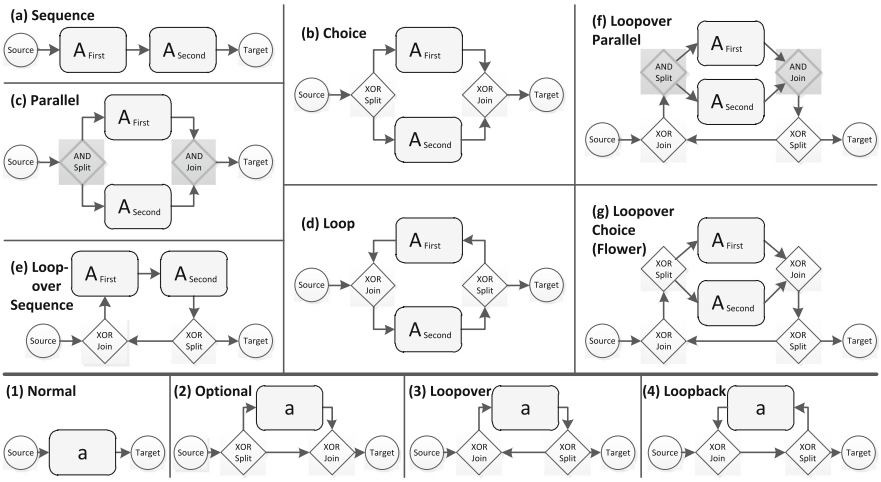

Fig. 2. Business process constructs supported by the CCM [15]

Algorithm 1. Methodology of the CCM in Pseudocode

```
   Data: Log L
   Result: BP bp
1  begin
2      A ← getSetOfAllActivitiesInLog(L);
3      BP bp ← buildInitialBPWithStartAndEnd();
4      bp ← getFootprintAndBuildConstruct(A, L, bp);
5      return bp;
6  Function getFootprintAndBuildConstruct(A_m, Log L, BP bp)
7      Footprint fp = extractFootprintForActivities(A_m, L);
8      if |A_m| = 1 then
9          Construct c ← analyseConstructForSingleActivity(fp);
10         bp ← createSingleActivityConstruct(c, A_m);
11     else
12         ConstructsSuitability[] cs ← calculateSuitabilityForConstructs(fp, A_m);
13         (Construct c, A_first, A_second) ← constructCompetition(cs, A_m);
14         bp ← createBlockConstruct(c, bp);
15         bp ← getFootprintAndBuildConstruct(A_first, L, bp);
16         bp ← getFootprintAndBuildConstruct(A_second, L, bp);
17     return bp;
```

standards like BPMN or YAWL; (3) If conflicting or exceptional behaviour exists in the log, the CCM picks the "best" fitting BP construct.

Algorithm 1 shows the conceptual methodology of the CCM algorithm in pseudocode. The CCM applies the divide-and-conquer paradigm and is implemented in a recursive fashion (see lines 7, 16, and 17). At the beginning `getFootprintAndBuildConstruct` is initially called for all involved activities ($A_m = A$) with the process bp consisting of only a start and end element. The recursive function is first creating a footprint fp from the given log L only considering the activities specified in set A_m (at the beginning all involved activities). In a next step it will be decided which is the best construct to represent the

behaviour captured by fp: (1) if the activity set A_m only consists of one element, it will be decided which of the single activity constructs (see bottom of Fig. 2) fits best - the process bp will then be enriched with the new single activity construct (see line 11); (2) If the activity set A_m contains more than one element, the suitability for each of the different constructs is calculated for any two activities $x, y \in A_m$ based on "soft" constraints and behaviour approximations, e.g. activities a and b are in a strong Sequence relationship. The result of this calculation (line 13) is a number of suitability matrices, one for each construct. In the subsequent competition algorithm it is determined what is the best combination of (A) the construct type $c \in \{Sequence, Choice, Loop, ...\}$, and (B) the two subsets A_{first} and A_{second} of A_m with $A_{first} \cup A_{second} = A_m$, $A_{first} \cap A_{second} = \{\}$, and $A_{first}, A_{second} \neq \{\}$, that best accommodate all x, y-pair relations of the corresponding matrix of construct c (line 14). The construct is then created and added to the existing process model bp (line 15), e.g. XOR-split and -join if the winning construct c was $Choice$. At this stage the recursive method calls will be executed to analyse and construct the respective behaviour for the subsets A_{first} and A_{second}. The split up of the set A_m continues in a recursive fashion until it cannot be divided any more, i.e. the set consists of a single activity (see case (1)). The process is completely constructed when the top recursive call returns.

Of particular interest for the transformation of the CCM algorithm to a solution for dynamic process discovery is the composition of the footprint and its calculation from the log. As opposed to many other process discovery algorithms, e.g. alpha-miner [20], the footprint does not consist of absolute relations, e.g. h is followed by a (see example in Fig. 1), but instead holds relative relation values, e.g. a is eventually followed by g in $0.4 \cong 40\,\%$ of the traces. Furthermore, the footprint only contains global relations between activities in order to guarantee a low polynomial execution time for the footprint interpretation [15]. The footprint of the CCM contains information about: (1) the occurrence of each involved activities $x \in A_m$, i.e. how many times x appears at least once per trace, how many times an x appears on average per trace, and how many times the trace started with x; (2) the global relations of each activity pair $x, y \in A_m$, i.e. in how many traces x appears sometime before the *first* occurrence of y in the trace, and in how many traces x appears sometime before *any* occurrence of y in the trace[2]. All measures in the footprint are relative to the number of traces in the log. Furthermore, not only one overall footprint is created for the CCM but also for every subset A_{first} and A_{second}, that is created during execution, a new sub-footprint is created (see Algorithm 1).

[2] This stands in contrast to existing discovery solutions since in the CCM the footprint and its interpretation is not based on *local* relationships between activity occurrences, e.g. direct neighbours, but based on *global* relationships between them.

5 Dynamic Constructs Competition Miner

As established in Sect. 1, increasingly dynamic processes and the need for immediate insight require current research in the domain of process mining to be driven by a set of additional challenges. To address these challenges the concept of Scalable Dynamic Process Discovery (SDPD), an interdisciplinary concept employing principles of CEP, Process Discovery, and EDBPM, has been introduced in [16]: "SDPD describes the method of monitoring one or more BPMSs in order to provide at any point in time a reasonably accurate representation of the current state of the processes deployed in the systems with regards to their control-flow, resource, and performance perspectives as well as the state of still open traces." That means, any potential changes in the mentioned aspects of the processes in the system that occur during run-time have to be recognized and reflected in the continuously updated "current state" of the process. Due to its purpose, for solutions of SDPD an additional set of requirements applies. For this paper, the most relevant of them are [16]:

- *Detection of Change*: An SDPD solution is required to detect change in two different levels defined in [17]: (1) Reflectivity: A change in a process instance (trace), i.e. every single event represents a change in the state of the associated trace. (2) Dynamism: A change on the business process level, e.g. because events/traces occurred that contradicts with the currently assumed process.
- *Algorithmic Run-Time*: An SDPD solution is applied as CEP concept and has to be able deal with large business processes operating with a high frequency, i.e. the actual run-time of the algorithms becomes very important. The key algorithms should be run-time effective to cope with increasing workload at minimal possible additional computational cost.

Motivated by these challenges the initial process discovery approach was altered to allow for dynamic process discovery. As opposed to the traditional *static* methodology (see Definition 2), *dynamic* process discovery is an iterative approach as defined in the following:

Definition 3. *Let log* $L_n = [e_0, e_1, ...e_n]$ *be a sequence of* $n+1$ *events ordered by time of occurrence (* $\forall i < j \wedge e_i, e_j \in L_n : time(e_i) \leq time(e_j)$*) and* BP_n *be the business process model representing the behaviour in* L_n*, then dynamic process discovery is defined as a function that projects the tuple* (e_n, BP_{n-1}) *to* BP_n*:*

$$DynamicProcessDiscovery : (e_n, BP_{n-1}) \Rightarrow BP_n$$

As described in Sect. 4, the CCM is a static mining algorithm and has to be modified in order to enable SDPD. The result of this modifications is called Dynamic CCM (DCCM). However, two restrictions for the DCCM with regards to the previously mentioned requirements of SDPD apply: (1) instead of discovering change on the BP perspectives control-flow, resources, and performance perspective, the DCCM described in this paper only focuses on discovering change in the control-flow, and (2) only change on the abstraction level of Dynamism

is detected, i.e. whether or not the control-flow of the process has changed - the detection of change on the abstraction level of Reflectivity will not be supported by the DCCM. Additionally to the requirements of SDPD the DCCM features the following important aspects: (1) *robust*: if conflicting, exceptional, or not representable behaviour occurs in the event stream, the DCCM does not fail but always picks the BP construct that best accommodates the recorded behaviour; (2) *deterministic*: the DCCM yields the exact same output BP for the same input stream of events.

Four different modifications were applied to the default CCM to create the DCCM. These modifications are summarised in the following list and described in more detail in the following sub-sections:

1. Splitting up the algorithm in two separate parts: one for dynamically updating the current footprint(s) complying to the requirement of extremely low algorithmic run-time, and one for interpreting the footprint into a BP model which has less restrictions with regards to its run-time.
2. In the CCM the footprint is calculated in relation to all occurring traces. This is not applicable for SDPD since the number of traces should not have an influence on the execution-time of any component of an SDPD solution. For this reason the footprint has to be calculated in a dynamic fashion, i.e. an event-wise footprint update independent from the previously occurred number of events or traces.
3. The original behaviour of the CCM to carry out a footprint calculation for every subset that has been created by the divide-and-conquer approach is not optimal as then the DCCM would have to extract up to $2 * n + 1$ different footprints if only one activity was split-up from the main set for each recursion.[3] This has been improved for the DCCM: for the most common constructs Choice and Sequence the sub-footprints are automatically derived from the parent footprint.
4. In rare cases it can happen that for every appearing event the state of the process is alternating between a number of different control-flows. This is caused by "footprint equivalent" BP models, i.e. two models are footprint equivalent if they both express the behaviour captured by the footprint. We introduce a measure which favours the last control-flow state in order to prevent the described behaviour.

5.1 Methodology of the Dynamic CCM

The original CCM algorithm had to be split up into two separate parts in order to comply to the SDPD's requirement of low algorithmic run-time for the event processing. A component triggered by the occurrence of a new event to update

[3] e.g. for $A = \{a, b, c, d\} : (a, b, c, d) \rightarrow ((a, b, c), (d)) \rightarrow (((a), (b, c)), (d)) \rightarrow (((a), ((b), (c))), (d))$, seven different footprints for sets $\{a, b, c, d\}, \{a, b, c\}, \{b, c\}, \{a\}, \{b\}, \{c\}, \{d\}$ need to be created - $(,)$ denote the nested blocks that emerge while splitting the sets recursively.

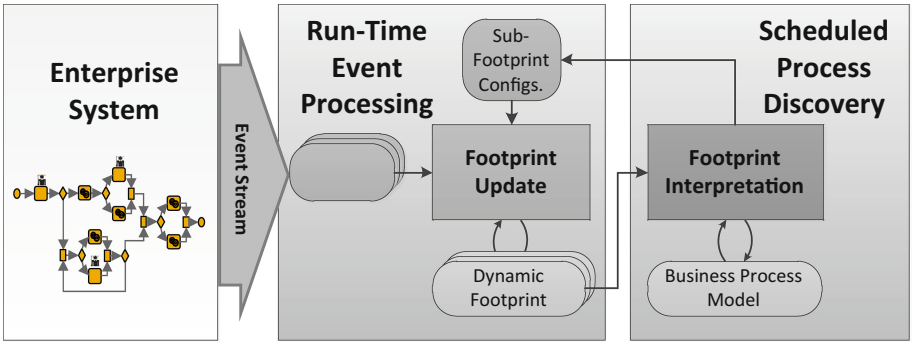

Fig. 3. Conceptual methodology of the dynamic CCM

the dynamic footprint and a component decoupled from the event processing which interprets the footprint into a BP Model. The conceptual methodology of the DCCM is depicted in Fig. 3. The components, models, and functionality of the DCCM are described in the following: Events from the monitored *Enterprise System*, in which the end-to-end process is deployed, are fed into an event stream. The *Footprint Update* component is the receiver of these events and processes them directly into changes on the overall *Dynamic Footprint* which represents the abstract state of the monitored business process. If additional footprints for subsets of activities are required as specified by the *Sub-Footprint Configurations*, e.g. if a Loop or Parallel construct was identified, then these sub-footprints are also updated (or created if they were not existent before). The *Dynamic Footprint(s)* can then at any point in time be compiled to a human-centric representation of the business process by the *Footprint Interpretation* component, i.e. the abstract footprint representation is interpreted into knowledge conforming to a block-structured BP model. In the DCCM this interpretation is scheduled dependent on how many new completed traces appeared, e.g. the footprint interpretation is executed once every 10 terminated traces. If the *interpretation frequency* $m \in \mathbb{N}$ of the DCCM is set to 1 a footprint interpretation is executed for every single trace that terminated. The *Footprint Interpretation* algorithm works similar to the CCM algorithm shown in Algorithm 1; but instead of extracting footprints from a log (line 8), the modified algorithm requests the readily available *Dynamic Footprint(s)*. If a sub-footprint is not yet available (e.g. at the beginning or if the process changed) the *Footprint Interpretation* specifies the request for a sub-footprint in the *Sub-Footprint Configurations* in the fashion of a feedback loop. Thus, *Sub-Footprint Configurations* and *Dynamic Footprints* act as interfaces between the two components, *Footprint Update* and *Footprint Interpretation*. The *Footprint Interpretation* cannot continue to analyse the subsets if no sub-footprint for these exist yet. In this case, usually occurring in the warm-up or transition phase, an intermediate BP model is created with activities containing all elements of the unresolved sets as depicted in Fig. 4.

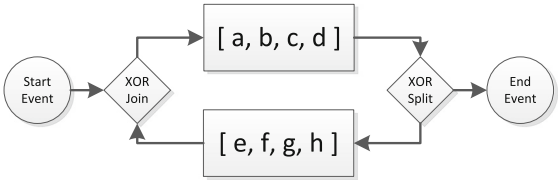

Fig. 4. Result of the *Footprint Interpretation* on an event stream produced by the example from Fig. 1 if no sub-footprints for $\{a, b, c, d\}$ and $\{e, f, g, h\}$ are available yet - only the top-level loop has been discovered

5.2 Run-Time Update of the Dynamic Footprint

The *Footprint Update* component processes events to changes in the *Dynamic Footprint*, i.e. updates the abstract representation of the process state. The original footprint extraction of the CCM algorithm calculates all values in relation to the number of occurred traces, i.e. every trace's influence on the footprint is equal: $\frac{1}{|traces|}$. To keep the algorithmic run-time to a minimum and allow for scalability the footprint update calculation should only take a fixed amount of time, independent from the total number of previously occurred events or traces. An increase of the total number of involved activities can cause, however, a linear increase of the execution-time due to the recalculation of the relations between the occurred activity and, in the worst case, all other activities. The independence from previous traces is the reason the footprint is calculated in a dynamic fashion, i.e. the dynamic footprint is incrementally updated in a way that older events "age" and thus have less influence than more recent events.

The general ageing approach that is utilized in the *Footprint Update* of the DCCM is based on the calculation of an individual *trace footprint*[4] (TFP) for each trace which influences the dynamic overall footprint (DFP). For the n-th new TFP_n the DFP is updated in the following way: Given a specified *trace influence factor* $t_{if} \in \mathbb{R}$ with $0 < t_{if} \leq 1$ the old DFP_{n-1} is aged by the *ageing factor* $a_f = 1 - t_{if}$, i.e.

$$DFP_n = t_{if} * TFP_n + (1 - t_{if}) * DFP_{n-1} \tag{1}$$

E.g., for trace influence factor $t_{if} = 0.01$: $DFP_n = 0.01 * TFP_n + 0.99 * DFP_{n-1}$. Two different ageing methods have been developed which will be evaluated against each other in Sect. 6: *Occurrence-based Ageing* and *Time-based Ageing*.

Occurrence-Based Ageing is an ageing method similar to the approach of Burretin et al. [5] discussed in Sect. 3. In this case the trace influence t_{if} is a fixed value and the DFP ages the same proportion for every time a trace footprint

[4] the occurrence values for activities as well as the global relations (see end of Sect. 4) are represented in the trace footprint by absolute statements $true \equiv 1$ if it occurred and $false \equiv 0$ if not.

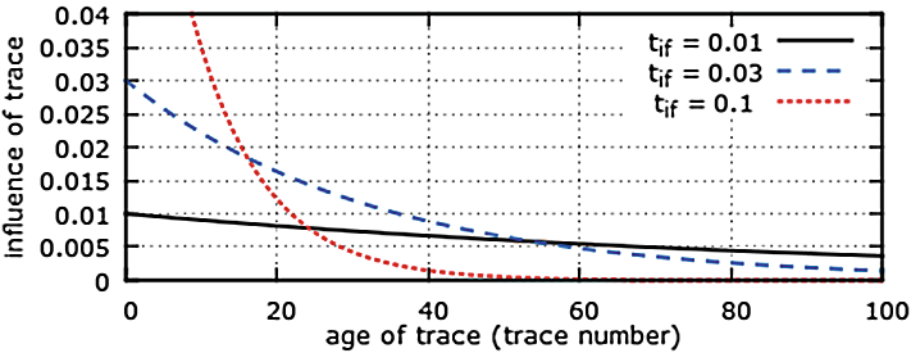

Fig. 5. Development of the influence of a trace for different trace influence factors(t_{if})

is added indeterminate from how much time has passed since the last footprint update. For example, assuming $t_{if} = 0.01$ the trace footprint TFP_n has the influence of 0.01 when it first occurs (see Eq. 1); after another TFP_{n+1} has occurred the influence of TFP_n decreases to $0.01 * 0.99$, and after another $0.01 * 0.99^2$ and so on. By applying this incremental method, older TFP are losing influence in the overall dynamic footprint. Figure 5 shows how the influence of a trace is dependent on its "age": If $t_{if} = 0.1$, the influence of a trace that appeared 60 traces ago became almost irrelevant. At the same time if $t_{if} = 0.01$ the influence of a trace of the same age is still a little more than half of its initial influence when it first appeared. Essentially, the purpose of the *trace influence factor* t_{if} is to configure both, the "memory" and the adaptation rate, of the footprint update component, i.e. a high t_{if} means quick adaptation but short memory but a small t_{if} means a slow adaptation but a long memory. Finding the correct trace influence is an issue of balancing these two inversely proportional effects, e.g. it might be generally desirable to have a high adaptation rate ($t_{if} = 0.1$) but if some behaviour of the process only occurs once in every 60 traces it will already be "forgotten" when it reappears (see Fig. 5) essentially resulting in a continuously alternating business process. However, while applying this method it was observed that at the beginning of the event streaming an unnecessarily long time to "warm-up" was required until the DFP reflected the correct behaviour of the business process. In order to shorten the "warm-up" phase of the *Footprint Update* a more dynamic method was adopted: If the overall amount of so far occurred traces $|traces| < \frac{1}{t_{if}}$ then the influence of the dynamic overall footprint is $\frac{|traces|}{|traces|+1}$ and of the new trace footprint $1 - \frac{1}{|traces|+1}$. As a result all traces that occur while $|traces| < \frac{1}{t_{if}}$ have the same influence in the DFP of $\frac{1}{|traces|}$. For instance if $t_{if} = 0.01$ and $|traces| = 9$ then a new dynamic footprint is calculated with $DFP_{10} = \frac{1}{10} * TFP + \frac{9}{10} * DFP_9$ and for the next trace $DFP_{11} = \frac{1}{11} * TFP + \frac{10}{11} * DFP_{10}$. As soon as $|traces| \geq \frac{1}{t_{if}}$ the standard occurrence ageing with a fixed influence factor is adopted:

$$DFP_n = \begin{cases} TFP_n & \text{if } n = 0 \\ \frac{1}{n} * TFP_n + \frac{n-1}{n} * DFP_{n-1} & \text{if } 0 < n < \frac{1}{t_{if}} \\ t_{if} * TFP_n + (1 - t_{if}) * DFP_{n-1} & \text{if } n \geq \frac{1}{t_{if}} \end{cases} \quad (2)$$

Because of this implementation the "warm-up" phase of the *Footprint Update* could be drastically reduced, i.e. processes were already completely discovered a few traces after the start of the monitoring which will be shown in Sect. 6.

Time-Based Ageing is an ageing method based on the time that has passed since the last trace occurred. The more time has passed the less influence the old DFP_{n-1} has on the updated DFP_n. This is achieved in a similar way than in the occurrence-based ageing but instead of having an ageing factor a_f relative to the trace occurrence it is now relative to the time passed. That means that ageing factor a_f and trace influence factor t_{if} are not fixed for each trace occurrence but calculated based on an *ageing rate* a_r per passed *time unit* t_{ur}, i.e. in particular time t_n has passed since the last trace occurred then

$$a_f = a_r^{\frac{t_n}{t_{ur}}} \text{ and } t_{if} = 1 - a_f$$

If $t_n = t_{ur}$ then the dynamic overall footprint ages exactly the same as with the occurrence-based ageing, if $t_n > t_{ur}$ then it ages quicker, and if $t_n < t_{ur}$ it ages slower. For instance, with ageing rate $a_r = 0.99$ and time unit $t_{ur} = 1s$: If the new trace occurred $t_n = 2s$ after the last footprint update then the new dynamic overall footprint $DFP_n = (1 - 0.99^2) * TFP_n + 0.99^2 * DFP_{n-1}$. Through time-based ageing the influence development of passed traces behaves similarly to the occurrence-based ageing in Fig. 5 apart from that the ageing is not based on trace-oldness but on time-oldness (the numbers on the x-axis now represent the passed time units since the trace occurred). For the time-based ageing a similar problem was observed during the warm-up phase than with the occurrence-based ageing: Since the DFP only consists of zeros when initialised it takes an unnecessary long time to converge towards a footprint representing the correct behaviour of the business process. For this reason an alternative linear ageing relative to the overall passed time since the first trace recorded t_{ur} was adopted as well. The final ageing factor a_f is the minimum of both calculated values as shown in Eq. 3.

$$a_f = \min(1 - \frac{t_n}{t_{all}}, a_r^{\frac{t_n}{t_{ur}}}) \quad (3)$$

$$DFP_n = (1 - a_f) * TFP_n + a_f * DFP_{n-1} \quad (4)$$

Considering the example from earlier where $a_r = 0.99$, time unit $t_{ur} = 1s$, the new trace TFP_n occurred $2s$ after the last update, and the first trace recorded was $t_{all} = 4s$ then $a_f = \min(1 - \frac{2s}{4s}, 0.99^2) = 0.5$ and according to Eq. 4: $DFP_n = (1-0.5) * TFP_n + 0.5 * DFP_{n-1}$. In this way the warm-up phase can be shortened similar to the occurrence-based approach but still be based on time.

Another important dynamism feature that had to be implemented was the possibility to add an activity that has not appeared before. A new activity is first recorded in the respective trace footprint. When the trace is terminated it will be added to the overall footprint in which it is not contained yet. The factored summation of both footprints to build the new dynamic footprint is carried out by assuming that a not previously in the dynamic overall footprint contained relation value is 0. Furthermore, activities that do not appear any more during operation should be removed from the dynamic footprint. This was implemented in the DCCM in the following way: If the *occurrence once* value of an activity drops below a removal threshold $t_r \in \mathbb{R}, t_r < t_{if}$ it will be removed from the dynamic footprint, i.e. all values and relations to other activities are discarded.

The fact that especially many Choice and Sequence constructs are present in common business processes, motivates an automated sub-footprint creation in the *Footprint Interpretation* based on the parent footprint rather then creating the sub-footprint from the event stream. This step helps to decrease the execution-time of the *Footprint Update* and was achieved by introducing an extra relation to the footprint[5] - the *direct neighbours* relation as used by other mining algorithms (see Sect. 3). In the *Footprint Interpretation* this relation is then used for creating the respective sub-footprints for Sequence and Choice constructs but not for identifying BP constructs since the *direct neighbours* relation does not represent a global relation between activities.

5.3 Modifications in the Footprint Interpretation Component

As analysed in the beginning of this section, the original behaviour of the CCM to retrieve a sub-footprint for each subset that has been created by the divide-and-conquer approach is not optimal. This is why, in the *Footprint Interpretation* the DCCM calculates the sub-footprints for the most common constructs, Choice and Sequence, from the available parent footprint: (1) For the Choice construct the probability of the exclusive paths are calculated with $p_{first} = \sum_{x \in A_{first}} Fel(x)$ and $p_{second} = \sum_{x \in A_{second}} Fel(x)$ with $Fel(x)$ being the occurrences of x as first element (see CCM footprint description in Sect. 4). Then the relevant values of the parent footprint are copied into their respective new sub-footprints and normalized, i.e. multiplied with $\frac{1}{p_{first}}$ and $\frac{1}{p_{second}}$, respectively. (2) The sub-footprints for the Sequence construct are similarly built, but without the normalization. Instead, the direct neighbours relation, now also part of the dynamic footprint, is used to calculate the new overall probabilities of the sub-footprints.

If two or more BP constructs are almost identically suitable for one and the same footprint, a slight change of the dynamic footprint might result in a differently discovered BP. This may cause an alternating behaviour for the footprint

[5] In rare cases (if Loop and Parallel constructs dominate) this modification can have a negative effect on the execution-time since extra information needs to be extracted without the benefit of mining less sub-footprints.

interpretation, i.e. with almost every footprint update the result of the interpretation changes. This is undesirable behaviour which is why the competition algorithm was additionally modified as follows: All combinations of BP construct and subsets are by default penalized by a very small value, e.g. $\frac{t_{tf}}{10}$, with the exception of the combination corresponding to the previously discovered BP model, hence reducing the risk of discovering alternating BP models.

6 Evaluation

The static CCM algorithm has been tested in detail for its accuracy in [15]: (1) in a qualitative analysis the CCM was able to rediscover 64 out of 67 processes for which a log was produced through simulation. (2) in the second part of the evaluation the discovery performance of the CCM was compared to the mining algorithms HeuristicsMiner (HM) [25], Inductive Miner (IM) [10], and the Flower Miner (FM), all of which are readily available in the ProM nightly build [21]. For ten given logs (including real-life logs and publicly available logs) the results of the algorithms (each configured with their default parameters) were evaluated for their *trace fitness* f_{tf}, *precision* f_{pr}, *generalization* f_g, and *simplicity* f_s with the help of the *PNetReplayer* plugin [24]. The averaged results of the detailed analysis are shown in Table 1 [15]; Note, that a lower simplicity value is better.

Table 1. Conformance results of the different discovery algorithms [15]

Trace Fitness f_{tf}				Precision f_{pr}				Generalization f_g				Simplicity f_s			
HM	IM	FM	CCM	HM	IM	FM	CCM	HM	IM	FM	CCM	HM	IM	FM	CCM
0.919	0.966	1.0	0.979	0.718	0.622	0.124	0.663	0.941	0.915	0.992	0.930	155.3	122.8	56.4	111.9

In the remainder of this section evaluation results of the DCCM are presented with regards to its capability of initially discovering the correct process and how it reacts to certain changes of a real-time monitored business process. Furthermore, a comparative analysis is carried out to determine the advantages and disadvantages of the two ageing strategies, occurrence-based and time-based ageing.

6.1 Experiment Setup

The experiments revolve around the concept of process rediscovery, i.e. that a source business process is executed and produces an event stream which is fed to the DCCM which then should discover a process behaviourally equivalent to the executed source process. Figure 6 shows three measures (t_w, t_d, and t_{tr}) which we use to evaluate the quality of the DCCM. In the figure BP_1 and BP_2 are the business processes deployed in the monitored system and BP'_1 to BP'_n are the models discovered by the DCCM. Additionally, BP_1 and BP'_m are equivalent ($BP_1 \equiv BP'_m$) as well as BP_2 and BP'_n ($BP_2 \equiv BP'_n$). For this part of the evaluation the following measures are of interest:

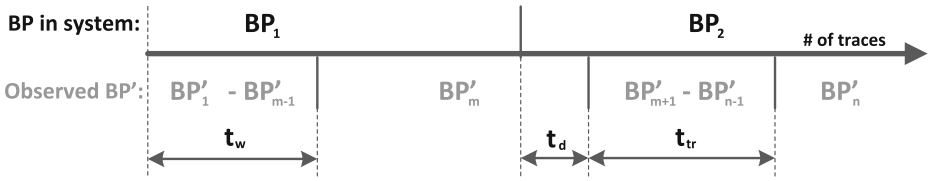

Fig. 6. Measures for detection of BP change in system

- Warm-up: $t_w \in \mathbb{N}$ the amount of completed traces the DCCM needs as input at the start until the resulting model equivalently represents the process in the system, i.e. until $BP_1 \equiv BP'_m$.
- Change Detection: $t_d \in \mathbb{N}$ the amount of completed traces it takes to detect a certain change in the monitored process - from the point at which the process changed in the system to the point at which a different process was detected. When the change is detected the newly discovered process is usually not equivalent to the new process in the system BP_2 but instead represents parts of the behaviour of both processes, BP_1 and BP_2.
- Change Transition Period: $t_{tr} \in \mathbb{N}$ the amount of completed traces it takes to re-detect a changed process - from the point at which the process change was detected to the point at which the correct process representation was identified, i.e. until $BP_2 \equiv BP'_n$. In this period multiple different business processes may be detected, each best representing the dynamic footprint at the respective point in time.

The basis of our evaluation is the example model in Fig. 1 which is simulated and the resulting event stream fed into the DCCM. In order to get reliable values for the three measures t_w, t_d, and t_{tr}, this is repeated 60 times for each configuration. From these 60 runs the highest and lowest 5 values for each measure are discarded and the average is calculated over the remaining 50 values. For each experiment the CCM core is configured with its default parameters (see [15]).

6.2 Warm-up Evaluation

The first experiment will evaluate how the DCCM behaves at the beginning when first exposed to the event stream, more particularly, we want to determine the duration of the warm-up phase t_w. Figure 7 shows the development of the first few BP models extracted by the DCCM using *Occurrence Ageing* with *trace influence factor* $t_{if} = 0.01$ (see Sect. 5.2) and *interpretation frequency* $m = 10$, i.e. an interpretation is executed every 10 completed traces: After the first trace the discovered process is a sequence reflecting the single trace that defines the process at that point in time. At trace 10, which is the next scheduled footprint interpretation, the algorithm discovers a Loop construct but cannot further analyse the subsets since the corresponding sub-footprint was not requested yet. Because of that, the feedback mechanism via the *Sub-Footprint Configurations* is utilized by the *Footprint Interpretation* algorithm to register the creation of

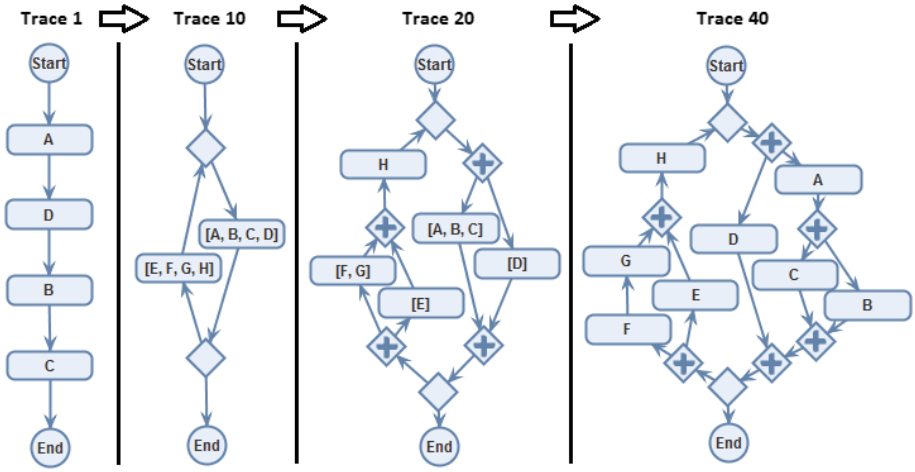

Fig. 7. The evolution of the discovered BP model during the warm-up phase

the missing sub-footprints. In the next scheduled run of the footprint interpretation, the Parallel construct of a, b, c, and d is discovered but again the analysis can not advance since a sub-footprint for the individual activity subsets has not been created yet. Activities e, f, g, and h seem to have appeared only in exactly this sequence until trace 20. Skipping one of the interpretation steps, we can see that at trace 40 the complete process has been mined, i.e. $t_w = 40$.

In Fig. 8 the development of warm-up duration t_w for of the DCCM with *Occurrence Ageing* for different $m \in \{1, 2, 3, 6, 10\}$ and $t_{if} \in \{0.001, 0.002, 0.005, 0.01, 0.018, 0.03\}$ is depicted. The warm-up phase seems generally very short and not strongly influenced by t_{if}. For $m = 10$ the warm-up phase cannot be any shorter because the example process consists of a block-depth of 3:

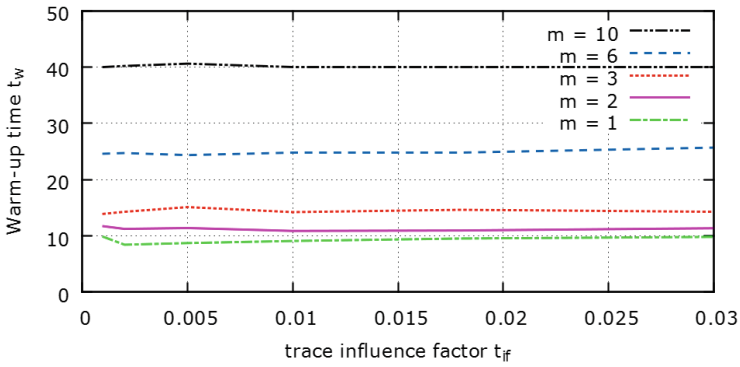

Fig. 8. Warm-up time with the occurrence ageing in relation to the trace influence factor

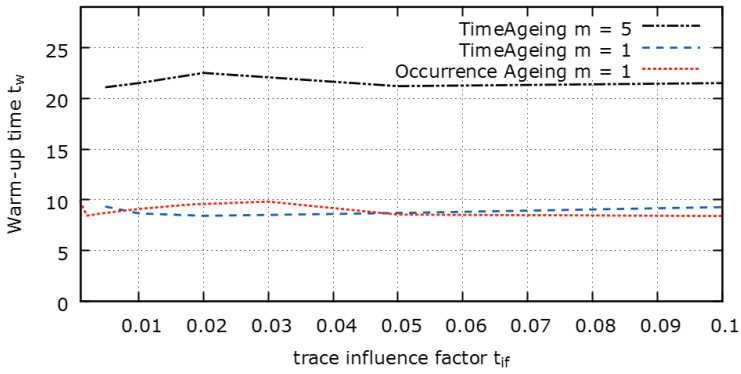

Fig. 9. Warm-up time with time ageing in relation to the trace influence factor

Parallel-in-Parallel-in-Loop, i.e. 3 subsequent requests for sub-footprints have to be made. This is an indicator that the modification effort to shorten the warm-up phase had a positive effect. No significant changes of t_w can be noticed when increasing or decreasing the *trace influence factor* t_{if}. This is caused by the optimisation rule that essentially nullifies the default ageing via t_{if} (see Eq. 2 in Sect. 5.2).

A similar sensitivity experiment was carried out for the warm-up duration t_w with the *Time-based Ageing* (Sect. 5.2): In order to make the results comparable the *ageing time unit* was set to one minute ($t_{ur} = 1min$) and the simulation produced on average approximately one instance per minute (instance occurrence: $1/min$; deviation: 0.5). As a result similar t_{if} for the occurrence-based and time-based[6] ageing should yield results of a similar magnitude and are thus comparable, i.e. 10 traces $\approx$ 10 min. In Fig. 9 the development of warm-up duration t_w for the DCCM with *Time Ageing* for different $m \in \{1, 5\}$ and $t_{if} \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$ is depicted. As a reference the results of the *Occurrence Ageing* with $m = 1$ were also added to the graph. Similar to the occurrence-based ageing, the development of the warm-up duration t_w seems to be in no relation to the trace influence factor t_{if} for the time-based ageing. This indicates that the optimisation rule for the time-based ageing successfully improves and nullifies the default ageing method for the warm-up phase (see Eqs. 3 and 4 in Sect. 5.2). Additionally, it can be seen that the result of the two different ageing types with a similar configuration yield similar results (for $m = 1$).

In order to examine the effect of the optimisations for both of the ageing methods, the same experiments were repeated without the respective optimisations (only for $m = 1$). Figure 10 shows that especially for the occurrence ageing a sizeable improvement was achieved through the application of the proposed optimisations, e.g. for $t_{if} = 0.02$ the non-optimized occurrence ageing took on average 341 traces until the model was rediscovered but the optimised version was already successful after 8.44 traces (on average). The non-optimised

[6] Time-based ageing is technically based on an *ageing rate* a_r rather than *trace influence factor* t_{if}. However, a_r can be derived from t_{if}, i.e. $a_r = 1 - t_{if}$.

time-based ageing already yielded comparatively good results which could be further improved by the optimisation proposed (see Eq. 3 in Sect. 5.2), e.g. for $t_{if} = 0.02$ the non-optimized time ageing took on average 37.16 min ($\approx$ number of traces) until the model was rediscovered but the optimised version was already successful after an average of 8.42 min.

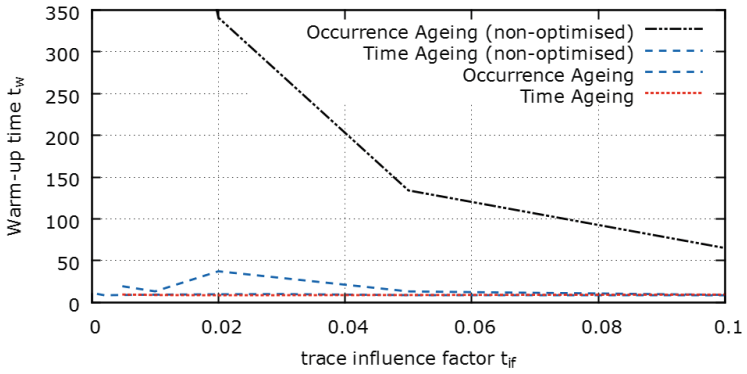

Fig. 10. Warm-up time without optimisation in relation to the trace influence factor with $m = 1$

6.3 Change Evaluation

In a second experiment we applied three changes of different extent to the business process (moving of an activity, adding an activity, and a complete process swap) during execution and are interested in the behaviour of the DCCM as well as in the change detection t_d and the change transition period t_{tr}.

Moving of Activity "A". The change applied is the move of activity A from the position before the inner Parallel construct to the position behind it. Figure 11 shows the evolution of the discovered BP models with occurrence ageing and *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$. The change was applied after 5753 traces. The footprint interpretation detects at the first chance to discover the change (trace 5760) a concept drift and finds via competition the best fitting construct: Parallel of a, c and b, d. The change detection t_d seemed to be unrelated to m and t_{if} for all experiment runs and was immediately recognized every time[7]. In Fig. 12 the development of t_{tr} for both ageing approaches different $m \in \{1, 10\}$ and $t_{if} \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$ is shown. For this change a clear difference between the performance of the occurrence-based and the time-based ageing can be observed: The time-based

[7] Note, that other changes like deletion of an activity will take longer to recognise, since their existence still "lingers" in the footprints "memory" for some time.

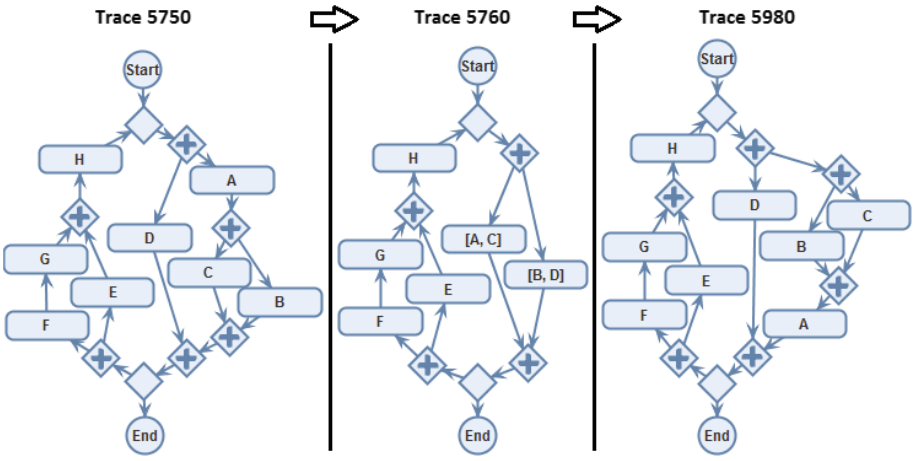

Fig. 11. The evolution of the discovered BP model during a change (move of activity "A")

ageing is significantly slower to finally detect the correct changed process, e.g. for $t_{if} = 0.02$ and $m = 1$ the time-based method detects the correct process BP_2 after on average 345 traces/minutes while the occurrence-based method recognises the new process correctly already after 105 traces (on average). Furthermore, it is observable that the change transition period t_{tr} was not particularly influenced by the interpretation frequency m but strongly influenced by t_{if}. If the value was very small ($t_{if} = 0.005$) a change took on average 450 traces (occurrence-based) or 1382 traces/minutes (time-based) in order to be reflected correctly in the discovered BP model. On the other hand if the trace influence factor is chosen very high, e.g. $t_{if} = 0.1$, the new process is correctly discovered after 21.1 (occurrence-based) or 67 (time-based) traces/minutes.

Adding of Activity "I". The change applied in this scenario is the addition of activity I at the end of the process. Figure 13 shows the evolution of the discovered BP models with occurrence ageing and *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$ while the change was applied after 5753 traces. It can be observed that the change detection t_d was again immediate, i.e. unrelated to m and t_{if}. However, it took on average 140 traces longer for the transition phase to be completed than for the previous scenario, i.e. on average 690 traces were necessary. The intermediate model which was valid from trace 5760 until 6450 (exclusively) recognises the relative position of activity I correctly (at the end of the process) but makes the activity optional. This is due to the fact that the "memory" of the dynamic overall footprint still contains behaviour from the original process in which the process ended without an activity I. Only after a certain amount of traces (dependent on the trace influence factor t_{if}) the memory of this behaviour became insignificant. Figure 14 shows an overview of the development of t_{tr} for all changes and both ageing approaches with $m = 1$ and $t_{if} \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$. Both ageing approaches behave similarly for the

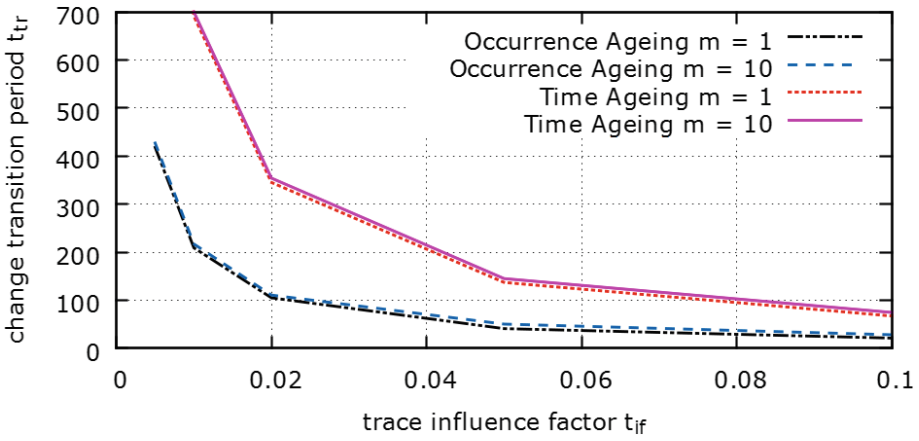

Fig. 12. The change transition period in relation to the trace influence factor (for recognising move of activity "A")

change of introducing the new activity I: even though the time-based ageing was usually slightly quicker, e.g. for $t_{if} = 0.01$: 687 (occurrence-based) vs. 684 (time-based) traces/minutes, the transition periods essentially only differed marginally (2 to 6 traces). When comparing the transition period t_{tr} for the occurrence-based method we can conclude that the movement of an activity was quicker detected correctly than the addition of a new activity ($t_{if} = 0.01$: 209 (move A) vs. 687 (add I)). However, for the time-based ageing only a relatively small difference was observed ($t_{if} = 0.01$: 691 (move A) vs. 684 (add I)).

Complex Change (Swap of Complete Process). The change applied in third scenario is a complete exchange of the original process during runtime, i.e. a revolutionary change. Figure 15 shows the evolution of the discovered BP models with occurrence ageing and *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$ with the change being applied after 5753 traces. The change detection t_d was again immediate (not displayed in Fig. 15), thus being unrelated to m and t_{if}. Many different process versions occurred during the transition phase of which only the version at trace 6310 is exemplary shown in the figure. Process BP_2 which is extremely different from BP_1 was finally correctly detected at trace 6680, i.e. on average 927 traces after the change was applied and 230 traces later than the introduction of a new activity (second scenario). When comparing the performance of occurrence-based and time-based ageing it can be observed that both develop similarly in relation to the trace influence factor t_{if} (see top two graphs in Fig. 14). This scenario can be considered the baseline scenario for how long a the transition phase may last at maximum for any t_{if}.

6.4 Occurrence-Based vs. Time-Based Ageing

A first observation is that the change detection t_d for all changes and ageing configuration was always quasi-immediate, i.e. whenever the first interpretation

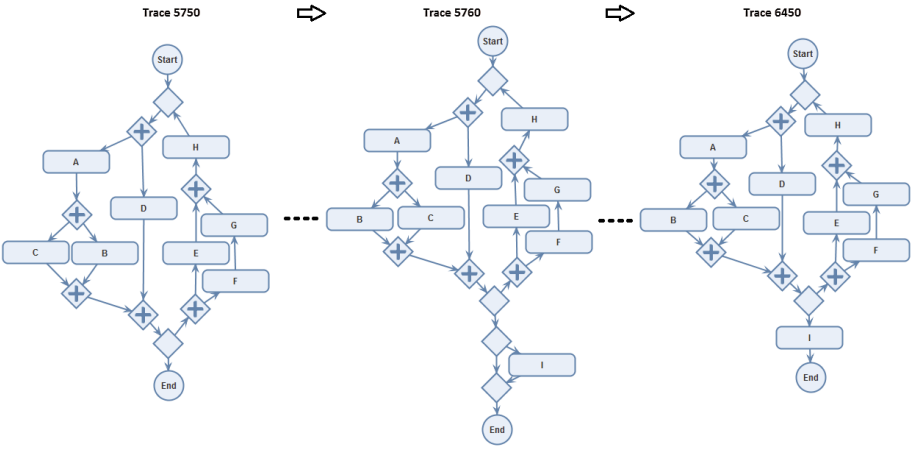

Fig. 13. The evolution of the discovered BP model during a change (addition of activity "I")

occurred after a change it was detected that the process has changed. This is mainly due to the configuration of the core CCM component and may change if different interpretation thresholds are selected. The ageing strategy, however, seems to not influence this directly (unless an extremely low trace influence t_{if} is selected).

A second observation is that there is a significant difference for the transition duration t_{tr} depending on the extent of the change, e.g. the movement of activity A took on average only $t_{tr} = 209$ traces to be correctly recognised whereas swapping process BP_1 with an entirely different process took on average $t_{tr} = 932$ to be recognised by the DCCM with occurrence-based ageing and $t_{if} = 0.01$ and $m = 1$.

Thirdly, time-based and occurrence-based ageing perform in most cases similarly well, with the exception of change scenario 1, in which activity A was moved to a different position within the process. Here, the occurrence-based ageing was able to detect the change significantly earlier than the time-based method, e.g. for $t_{if} = 0.1$ the new process is correctly discovered after 21.1 (occurrence-based) or 67 (time-based) traces/minutes. However, a fact not shown in the graphs is the number of exceptional results: While the time-based ageing did not suffer any exceptional experiment results it was observed that in approximately 4 % of the experiments for the occurrence-based method disproportionately high values for t_{tr} occurred. Due to the experiment setup, these results were ignored (see Sect. 6.1 - removal of the five highest and lowest values). In this context it was furthermore observed that a very high trace influence $t_{if} >> 0.1$ (not part of the above experiments) resulted in frequently changing/alternating discovered BP models even though the source process producing the events did not change. The reason for this behaviour is that not all variations of the process are included in the current dynamic footprint because they have already been

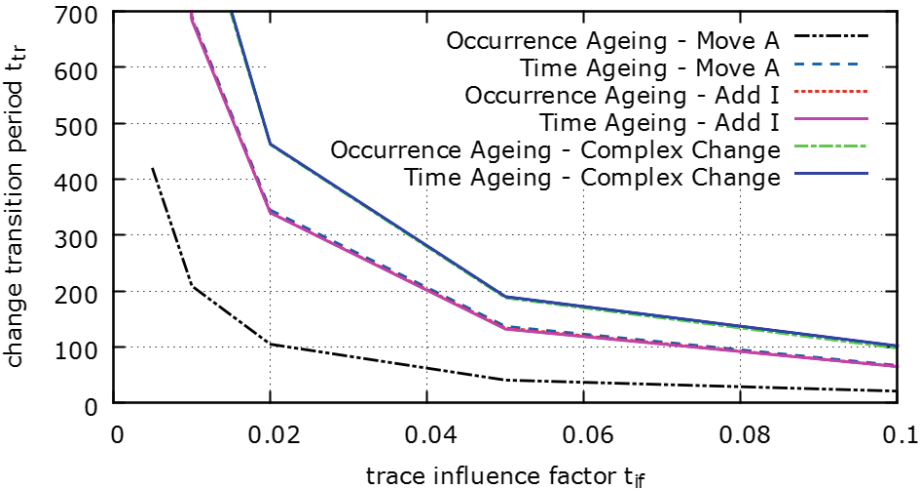

Fig. 14. The change transition period in relation to the trace influence factor for all three changes (m = 1)

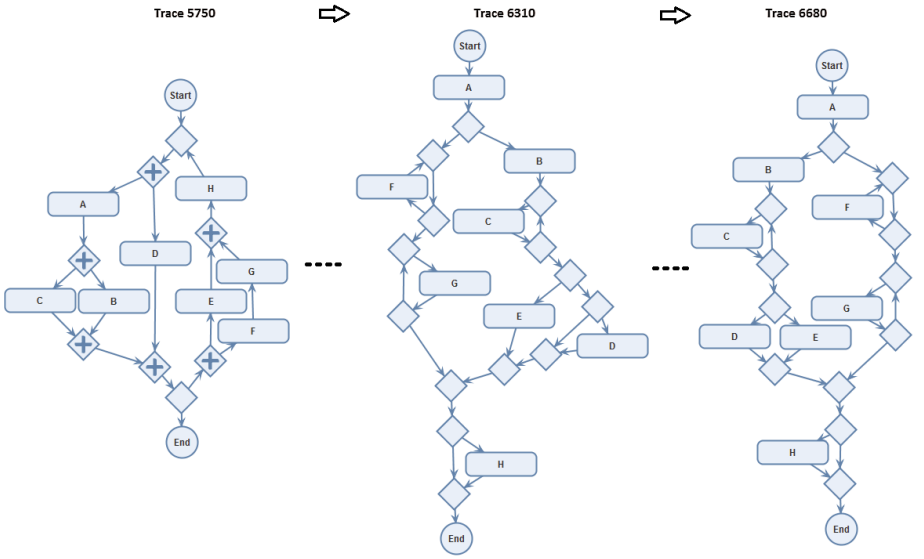

Fig. 15. The evolution of the discovered BP model during a change (complex change)

"forgotten" before they reappeared. It was observed that for the examined scenarios the occurrence-based ageing is slightly more susceptible to this issue than the time-based approach. Generally, the issue of "forgetting" too quickly is more likely to occur in large business processes containing rarely executed but still relevant behaviour and emphasises the importance of setting the trace influence

factor t_{if} correctly to balance between timely correct discovery (higher t_{if}) and a sufficiently long memory (lower t_{if}) to not forget frequently occurring behaviour.

Additionally to the warm-up and change evaluation, first performance tests have been carried out for large artificially produced processes. For a randomly created and strongly nested process consisting of 100 activities the throughput of the footprint update was close to 100,000 events per second and the footprint interpretation successfully discovered the process in a matter of seconds. Although not tested yet in a real-life setting, the shown results indicate that the DCCM is very suitable for discovering and monitoring large enterprise processes.

7 Conclusion and Future Work

In this paper we suggested modifications for the Constructs Competition Miner to enable Scalable Dynamic Process Discovery as proposed in [16]. The CCM is a process discovery algorithm that follows a divide-and-conquer approach to directly mine a block-structured process model which consists of common BP-domain constructs and represents the main behaviour of the process. This is achieved by calculating global relations between activities and letting the different supported constructs compete with each other for the most suitable solution from top to bottom using "soft" constraints and behaviour approximations. The CCM was designed to deal with noise and not-supported behaviour. To apply the CCM in a real-time environment it was split up into two separate parts, executed on different occasions: (1) the *footprint update* which is called for every occurring event and updates the dynamic footprint(s) and (2) the footprint interpretation which derives the BP model from the dynamic footprint through applying a modified top-down competition approach of the original CCM algorithm. The modifications on the CCM were mostly motivated by the objective to keep algorithmic run-time of the individual algorithms to a minimum. This was successfully implemented which is shown by the performance results in the evaluation section. Both possible ageing methods, occurrence-based and time-based ageing, showed reasonably good results, especially with the optimisations to reduce the warm-up duration t_w. It was furthermore shown that changes in the monitored process are almost instantly detected, i.e. $t_d \approx 0$.

The presented approach of Dynamic CCM (DCCM) is driven by the requirements of real life industrial use cases provided by business partners within the EU funded project TIMBUS. During the evaluation in the context of the use-cases it became apparent that this concept still has a number of limitations which are considered to be future work: (1) Changes in the state of the business process are usually detected almost immediately but it may take a long time until the new state of the system is reflected appropriately in the extracted business process model. This behaviour originates from the fact that the footprint and the interpreted business process are in a sort of intermediate state for a while until the influence of the old version of the business process has disappeared. Furthermore, the trace influence factor t_{if} is a pre-specified value but in reality it is dependent on how many traces

we need to regard to represent all the "behaviour" of the model[8]. This in turn is strongly dependent on the amount of activities in the model, since more activities usually mean more control-flow behaviour. A possible future modification could be to have the influence factor dynamically adapt, i.e. similar to the self-adapting ageing proposed in [5]. (2) If no sub-footprint is available for a set of activities, the footprint interpreter does not further analyse this set. Through approximations or the use of the direct neighbours relation at least a "close enough" control-flow for the subset could be retrieved. (3) The discovery of the state of a business process should also comprise information of other perspectives than the control-flow, e.g. resource and performance.

References

1. von Ammon, R., Ertlmaier, T., Etzion, O., Kofman, A., Paulus, T.: Integrating complex events for collaborating and dynamically changing business processes. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 370–384. Springer, Heidelberg (2010)
2. Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I., 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)
3. Cattafi, M., Lamma, E., Riguzzi, F., Storari, S.: Incremental declarative process mining. In: Szczerbicki, E., Nguyen, N.T. (eds.) Smart Information and Knowledge Management. SCI, vol. 260, pp. 103–127. Springer, Heidelberg (2010)
4. Buijs, J., Van Dongen, B., Van Der Aalst, W.: A genetic algorithm for discovering process trees. In: Evolutionary Computation (CEC), pp. 1–8. IEEE (2012)
5. Burattin, A., Sperduti, A., Van Der Aalst, W.: Heuristics miners for streaming event data. CoRR abs/1212.6383 (2012)
6. Kindler, E., Rubin, V., Schäfer, W.: Incremental workflow mining based on document versioning information. In: Li, M., Boehm, B., Osterweil, L.J. (eds.) SPW 2005. LNCS, vol. 3840, pp. 287–301. Springer, Heidelberg (2006)
7. Kindler, E., Rubin, V., Schäfer, W.: Activity mining for discovering software process models. In: Software Engineering 2006, Fachtagung des GI-Fachbereichs Softwaretechnik, LNI, pp. 175–180. GI (2006)
8. Ko, R.K.L.: A computer scientist's introductory guide to business process management (BPM). ACM Crossroads J. **15**(4), 11–18 (2009)
9. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013 Workshops. LNBIP, vol. 171, pp. 66–78. Springer, Heidelberg (2014)
11. Luckham, D.: The Power of Events: An Introduction to Complex Event Processingin Distributed Enterprise Systems. Addison-Wesley Professional, Reading (2002)

[8] if t_{if} is set too high normal behaviour unintentionally becomes exceptional behaviour.

12. Maggi, F.M., Burattin, A., Cimitile, M., Sperduti, A.: Online process discovery to detect concept drifts in LTL-based declarative process models. In: Meersman, R., Panetto, H., Dillon, T., Eder, J., Bellahsene, Z., Ritter, N., Leenheer, P., Dou, D. (eds.) ODBASE 2013. LNCS, vol. 8185, pp. 94–111. Springer, Heidelberg (2013)
13. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, pp. 346–357. Morgan Kaufmann (2002)
14. OMG Inc.: Business Process Model and Notation (BPMN) Specification 2.0. formal/2011-01-03 (2011). http://www.omg.org/spec/BPMN/2.0/PDF
15. Redlich, D., Molka, T., Gilani, W., Blair, G., Rashid, A.: Constructs competition miner: process control-flow discovery of BP-domain constructs. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 134–150. Springer, Heidelberg (2014)
16. Redlich, D., Gilani, W., Molka, T., Drobek, M., Rashid, A., Blair, G.: Introducing a framework for scalable dynamic process discovery. In: Aveiro, D., Tribolet, J., Gouveia, D. (eds.) EEWC 2014. LNBIP, vol. 174, pp. 151–166. Springer, Heidelberg (2014)
17. Redlich, D., Blair, G., Rashid, A., Molka, T., Gilani, W.: Research challenges for business process models at run-time. In: Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (eds.) Models@run.time. LNCS, vol. 8378, pp. 208–236. Springer, Heidelberg (2014)
18. Solé, M., Carmona, J.: Incremental process mining. In: Proceedings of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2010) and of the 10th International Conference on Application of Concurrency to System Design (ACSD 2010). CEURWorkshop Proceedings, pp. 175190. CEUR-WS.org (2010)
19. Van Der Aalst, W., Ter Hofstede, A.: YAWL: Yet Another Workflow Language (2003)
20. Van Der Aalst, W., Weijters, A., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
21. Van Der Aalst, W., Van Dongen, B.: ProM: the process mining toolkit. Ind. Eng. **489**, 1–4 (2009)
22. 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)
23. Van Der Aalst, W.: Process Mining - Discovery, Conformance and Enhancementof Business Processes. Springer, Heidelberg (2011)
24. Van Der Aalst, W., Adriansyah, A., Van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
25. Weijters, A., Van Der Aalst, W., Alves de Medeiros, A.: Process mining with the heuristics miner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology (2006)