


# Discovering workflow nets using integer linear programming

S. J. van Zelst<sup>1</sup>  · B. F. van Dongen<sup>1</sup> ·  
W. M. P. van der Aalst<sup>1</sup> · H. M. W. Verbeek<sup>1</sup>

Received: 19 July 2016 / Accepted: 25 October 2017  
© The Author(s) 2017. This article is an open access publication

**Abstract** Process mining is concerned with the analysis, understanding and improvement of business processes. Process discovery, i.e. discovering a process model based on an event log, is considered the most challenging process mining task. State-of-the-art process discovery algorithms only discover local control flow patterns and are unable to discover complex, non-local patterns. Region theory based techniques, i.e. an established class of process discovery techniques, do allow for discovering such patterns. However, applying region theory directly results in complex, overfitting models, which is less desirable. Moreover, region theory does not cope with guarantees provided by state-of-the-art process discovery algorithms, both w.r.t. structural and behavioural properties of the discovered process models. In this paper we present an ILP-based process discovery approach, based on region theory, that guarantees to discover relaxed sound workflow nets. Moreover, we devise a filtering algorithm, based on the internal working of the ILP-formulation, that is able to cope with the presence of infrequent, exceptional behaviour. We have extensively evaluated the technique using different event logs with different levels of exceptional behaviour. Our experiments show that the presented approach allows us to leverage the inherent shortcomings of existing region-based approaches. The techniques presented are implemented and

---

✉ S. J. van Zelst  
s.j.v.zelst@tue.nl

B. F. van Dongen  
b.f.v.dongen@tue.nl

W. M. P. van der Aalst  
w.m.p.v.d.aalst@tue.nl

H. M. W. Verbeek  
h.m.w.verbeek@tue.nl

<sup>1</sup> Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

readily available in the HybridILPMiner package in the open-source process mining tool-kits ProM (<http://promtools.org>) and RapidProM (<http://rapidprom.org>).

**Keywords** Process mining · Process discovery · Region theory · Integer linear programming

**Mathematics Subject Classification** 97R50 · 90C05

## 1 Introduction

The execution of business processes within a company generates traces of event data in its supporting information system. The goal of *process mining* [28] is to turn this data, recorded in *event logs*, into actionable knowledge. Three core branches form the basis of process mining: *process discovery*, *conformance checking* and *process enhancement*. In process discovery, this paper's focus, the goal is to construct a process model based on an event log. In conformance checking the goal is to assess whether a given process model and event log conform with respect to each other in terms of described behaviour. In process enhancement the goal is to improve processes models, primarily, though not exhaustively, using the two aforementioned fields.

Several *different process models* exist that (largely) describe the behaviour in an event log. Hence, we need means to rank and compare these different process models. In process mining we typically judge the quality of process models based on four essential quality dimensions: *replay-fitness*, *precision*, *generalization* and *simplicity* [7, 28, 33]. Replay-fitness describes the fraction of behaviour in the event log that is also described by the model. Precision describes the fraction of behaviour described by the model that is also present in the event log. Generalization indicates a model's ability to account for behaviour not part of the event log, e.g. in case of parallelism, it is often impossible to observe all possible behaviour in the event log. Simplicity refers to a model's interpretability by a human analyst. A process discovery result ideally strikes an adequate balance between these four quality dimensions.

State-of-the-art process discovery algorithms guarantee that their discovered process models have both structural and behavioural properties [6, 16]. These guarantees have a positive impact on the aforementioned quality dimensions. As a consequence, the techniques are unable to find complex, non-local control flow patterns [29]. In [34] an integer linear programming (ILP) [24] based process discovery algorithm is proposed that is able to find such patterns. However, the algorithm does not provide the same guarantees as most state-of-the-art process discovery algorithms. Moreover, the algorithm only works well under the assumption that the event log only holds frequent behaviour that fits nicely into some underlying process model.

Real event logs typically include low-frequent exceptional behaviour, e.g. caused by people deviating from the normative process or cases that require special treatment. Because of this, applying ILP-based process discovery as-is on real data often yields, despite its potential, unsatisfactory results. In this paper we present a revised ILP-based process discovery algorithm that solves the inherent shortcomings of current ILP-based approaches. Our contribution is summarized as follows: (1) We show that our approach is able to discover *relaxed sound workflow nets*, and (2) We present an

effective, integrated, filtering algorithm that results in process models that abstract from infrequent and/or exceptional behaviour.

The proposed algorithm is implemented in the process mining framework PROM [39] (*HybridILPMiner* package) and available in RapidPROM [5, 26]. We have compared our technique with two state-of-the-art filtering techniques [9, 15]. We additionally validated the applicability of our approach on two real life event logs [11, 19]. Our experiments confirm the effectiveness of the proposed approach, both in terms of resulting model quality and computational complexity.

The remainder of this paper is organized as follows. In Sect. 2, we discuss related work. In Sect. 3, we present background concepts. In Sect. 4, we show how to discover relaxed sound workflow nets. In Sect. 5, we present an integrated filtering algorithm that eliminates infrequent behaviour. In Sect. 6, we evaluate the proposed approach. Sect. 7 concludes the paper.

## 2 Related work

In this section we predominantly focus on the application of *region theory*, i.e. the theoretical foundation of ILP-based process discovery, in process discovery. We additionally discuss filtering techniques designed for process discovery.

### 2.1 Process discovery

The state-of-the-art process discovery algorithm, i.e. the Inductive Miner [16], discovers process models by applying a divide-and-conquer approach. The algorithm splits the event log into smaller parts and, recursively, finds models for these sub-logs which are later combined. The resulting models are hierarchically structured sound workflow nets [27]. A limitation of the approach is its inability to discover complex non-local control flow patterns. The discovery approach presented in [6], i.e. the Evolutionary Tree Miner, is able to find similar process models as the Inductive Miner. The algorithm opts an evolutionary computational approach and is therefore non-deterministic and does not guarantee termination. Like the Inductive Miner, it is not able to discover complex non-local control flow patterns. For an overview of other process discovery algorithms we refer to [12, 28, 35].

Several process discovery techniques are proposed based on region theory, i.e. a solution to the Petri net synthesis problem [23]. Region theory comes in two forms, i.e. state-based region theory [4, 13, 14] using transition systems as an input and language-based region theory [1, 2, 10, 17, 18] using languages as an input. The main difference between the synthesis problem and process discovery is related to generalization of the discovered models. Process models found by classical region theory approaches have perfect replay-fitness and maximal precision. Process discovery on the other hand aims at extracting a generalizing process model, i.e. precision, and in some cases replay-fitness, need not be maximized.

In [31] a process discovery approach is presented that transforms an event log into a transition system, after which state-based region theory is applied. Constructing the transition system is strongly parametrized, i.e. using different parameters yields dif-

ferent process discovery results. In [25] a similar approach is presented. The main contribution is a complexity reduction w.r.t. conventional region-based techniques. In [3] a process discovery approach is presented based on language-based region theory. The method finds a minimal linear basis of a polyhedral cone of integer points, based on the event log. It guarantees perfect replay-fitness, whereas it does not maximize precision. The worst-case time complexity of the approach is exponential in the size of the event log. In [8] a process discovery algorithm is proposed based on the concept of numerical abstract domains. Based on the event log's prefix-closure, a convex polyhedron is approximated by means of calculating a convex hull. The convex hull is used to compute causalities in the input event log by deducing a set of linear inequalities which represent places. In [34] a first design of a process discovery ILP-formulation is presented. An objective function is presented, which is generalized in [37], that allows for expressing a preference for finding certain Petri net places. The work also presents means to formulate ILP constraints that help finding more advanced Petri net-types, e.g. Petri nets with reset- and inhibitor arcs.

All aforementioned techniques leverage the strict implications of region theory w.r.t. process discovery, i.e. precision maximization, poor generalization and poor simplicity, to some extent. However, the techniques still perform suboptimal. Since the techniques guarantee perfect replay-fitness, they tend to fail if exceptional behaviour is present in the event log, i.e. they produce models that are incorporating infrequent behaviour (outliers).

## 2.2 Filtering infrequent behaviour

Little work has been done regarding filtering of infrequent behaviour in context of process mining. The majority of work concerns unpublished/undocumented ad-hoc filtering implementations in the ProM framework [39].

In [9] an event log filtering technique is presented that filters on *event level*. Events within the event log are removed in case they do not fit an underlying, event log based, automaton. The technique can be used as a *pre-processing* step prior to invoking any discovery algorithm. In [15] the Inductive Miner [16] is extended with filtering capabilities to handle infrequent behaviour. The technique is tailored towards the internal working of the Inductive Miner algorithm and considers three different types of filters. Moreover, the technique exploits the inductive nature of the underlying algorithm, i.e. filters are applied on multiple levels.

## 3 Background

In this section we present basic notational conventions, event logs and workflow nets. Moreover, we present a process discovery ILP-formulation based on [34,37].

### 3.1 Bags, sequences and vectors

$X = \{e_1, e_2, \dots, e_n\}$  denotes a set.  $\mathcal{P}(X)$  denotes the power set of  $X$ .  $\mathbb{N}$  denotes the set of positive integers *including* 0 whereas  $\mathbb{N}^+$  *excludes* 0.  $\mathbb{R}$  denotes the set of

real numbers. A bag (multiset) over  $X$  is a function  $B : X \rightarrow \mathbb{N}$  which we write as  $[e_1^{v_1}, e_2^{v_2}, \dots, e_n^{v_n}]$ , where for  $1 \leq i \leq n$  we have  $e_i \in X, v_i \in \mathbb{N}^+$  and  $B(e_i) = v_i$ . If for some element  $e, B(e) = 1$ , we omit its superscript. An empty bag is denoted as  $\emptyset$ . Element inclusion applies to bags: if  $e \in X$  and  $B(e) > 0$  then also  $e \in B$ . Set operations, i.e.  $\cup, \setminus, \cap$ , extend to bags. The set of all bags over  $X$  is denoted  $\mathcal{B}(X)$ .

A sequence  $\sigma$  of length  $k$  relates positions to elements  $e \in X$ , i.e.  $\sigma : \{1, 2, \dots, k\} \rightarrow X$ . An empty sequence is denoted as  $\epsilon$ . We write every non-empty sequence as  $\langle e_1, e_2, \dots, e_k \rangle$ . The set of all possible sequences over a set  $X$  is denoted as  $X^*$ . We write *concatenation* of sequences  $\sigma_1$  and  $\sigma_2$  as  $\sigma_1 \cdot \sigma_2$ , e.g.  $\langle a, b \rangle \cdot \langle c, d \rangle = \langle a, b, c, d \rangle$ . Let  $Y \subseteq X$ , we define  $\downarrow_Y : X^* \rightarrow Y^*$  recursively with  $\downarrow_Y(\epsilon) = \epsilon$  and  $\downarrow_Y(\langle x \rangle \cdot \sigma) = \langle x \rangle \cdot \downarrow_Y(\sigma)$  if  $x \in Y$  and  $\downarrow_Y(\sigma)$  otherwise. We write  $\sigma_{\downarrow_Y}$  for  $\downarrow_Y(\sigma)$ .

Given  $Y \in X^*$ , the *prefix-closure* of  $Y$  is:  $\bar{Y} = \{\sigma_1 \in X^* | \exists \sigma_2 \in X^* (\sigma_1 \cdot \sigma_2 \in Y)\}$ . We extend the notion of a prefix-closure on bags of sequences. Let  $Y \subseteq X^*$  and  $B_Y : Y \rightarrow \mathbb{N}$  we define  $\bar{B}_Y : \bar{Y} \rightarrow \mathbb{N}$ , such that:  $\bar{B}_Y(\sigma) = B_Y(\sigma) + \sum_{\sigma \cdot \langle e \rangle \in \bar{Y}} \bar{B}_Y(\sigma \cdot \langle e \rangle)$ . For example,  $B_2 = [\langle a, b \rangle^5, \langle a, c \rangle^3]$  yields  $\bar{B}_2 = [\epsilon^8, \langle a \rangle^8, \langle a, b \rangle^5, \langle a, c \rangle^3]$ .

Given set  $X$  and a range of values  $R \subseteq \mathbb{R}$ . Vectors are denoted as  $\mathbf{z} \in R^{|X|}$ , where  $\mathbf{z}(e) \in R$  and  $e \in X$ . We assume vectors to be *column vectors*. For vector multiplication we assume that vectors agree on their indices. Throughout the paper we assume a *total ordering on sets of the same domain*. Given  $X = \{e_1, e_2, \dots, e_n\}$  and  $\mathbf{z}_1, \mathbf{z}_2 \in R^{|X|}$  we have  $\mathbf{z}_1^T \mathbf{z}_2 = \sum_{i=1}^n \mathbf{z}_1(e_i) \mathbf{z}_2(e_i)$ . A *Parikh vector*  $\mathbf{p}$  represents the number of occurrences of an element within a sequence, i.e.  $\mathbf{p} : X^* \rightarrow \mathbb{N}^{|X|}$  with  $\mathbf{p}(\sigma) = (\#_{e_1}(\sigma), \#_{e_2}(\sigma), \dots, \#_{e_n}(\sigma))$  where  $\#_{e_i}(\sigma) = |\{i' \in \{1, 2, \dots, |\sigma|\} | \sigma(i') = e_i\}|$ .

### 3.2 Event logs and workflow nets

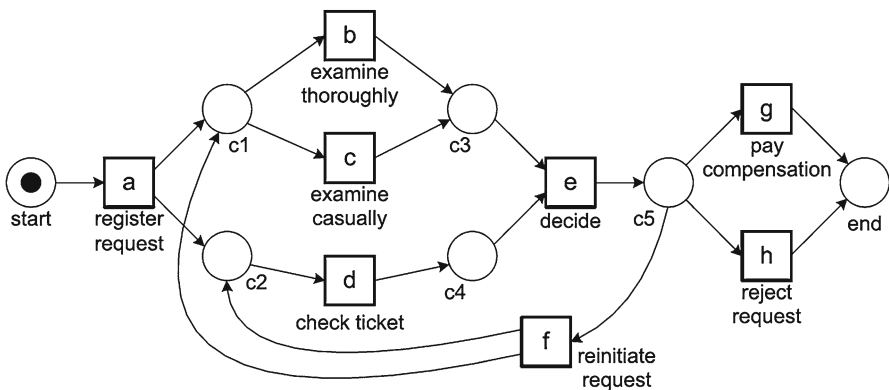
In process discovery, event logs, which describe the actual execution of activities in context of a business process, are the main source of input. An example event log is presented in Table 1. Consider all activities related to *Case-id 1*. John registers a request, after which Lucy examines it thoroughly. Pete checks the ticket after which Rob decides to reject the request. The execution of an activity in context of a business process is referred to as an *event*. A sequence of events, e.g. the sequence of events related to case 1, is referred to as a *trace*.

Let  $\mathcal{A}$  denote the universe of all possible activities. An event log  $L$  is a bag of sequences over  $\mathcal{A}$ , i.e.,  $L \in \mathcal{B}(\mathcal{A}^*)$ . Typically, there exists  $A_L \subset \mathcal{A}$  of activities that are actually present in  $L$ . In some cases we refer to an event log as  $L \in \mathcal{B}(A_L^*)$ . A sequence  $\sigma \in L$  represents a trace. We write case 1 as trace  $\langle \text{“register request”, “examine thoroughly”, “check ticket”, “decide”, “reject request”} \rangle$ . In the remainder of the paper, we use simple characters for activity names, e.g. we write case 1 as  $\langle a, b, d, e, h \rangle$ .

The goal within process discovery is to discover a process model based on an event log. In this paper we consider *workflow nets (WF-nets)* [27], based on *Petri nets* [22], to describe process models. We first introduce Petri nets and their execution semantics, after which we define workflow nets.

**Table 1** Fragment of a fictional event log [28] (a row corresponds to an event)

Case-id	Activity	Resource	Time-stamp
...	...	...	...
1	Register request ( <i>a</i> )	John	2015-05-08:08.45
1	Examine thoroughly ( <i>b</i> )	Lucy	2015-05-08:09.13
2	Register request ( <i>a</i> )	John	2015-05-08:09.14
2	Check ticket ( <i>d</i> )	Pete	2015-05-08:10.11
1	Check ticket ( <i>d</i> )	Pete	2015-05-08:10.28
2	Examine causally ( <i>b</i> )	Rob	2015-05-08:10.43
1	Decide ( <i>e</i> )	Rob	2015-05-08:11.14
1	Reject request ( <i>h</i> )	Rob	2015-05-08:11.35
...	...	...	...



**Fig. 1** Example WF-net  $W_1$ , adopted from [28]

A Petri net is a *bipartite graph* consisting of a set of vertices called *places* and a set of vertices called *transitions*. Arcs connect places with transitions and vice versa. Additionally, transitions have a (possibly unobservable) label which describes the activity that the transition represents. A Petri net is a quadruple  $N = (P, T, F, \lambda)$ , where  $P$  is a set of places and  $T$  is a set of transitions with  $P \cap T = \emptyset$ .  $F$  denotes the flow relation of  $N$ , i.e.,  $F \subseteq (P \times T) \cup (T \times P)$ .  $\lambda$  denotes the label function, i.e. given a set of labels  $\Lambda$  and a symbol  $\tau \notin \Lambda$ , it is defined as  $\lambda: T \rightarrow \Lambda \cup \{\tau\}$ . For a node  $x \in P \cup T$ , the pre-set of  $x$  in  $N$  is defined as  $\bullet x = \{y \mid (y, x) \in F\}$  and  $x \bullet = \{y \mid (x, y) \in F\}$  denotes the post-set of  $x$ . Graphically we represent places as *circles* and transitions as *boxes*. For every  $(x, y) \in F$  we draw an *arc* from  $x$  to  $y$ . An example Petri net (which is also a WF-net) is depicted in Fig. 1. Observe that we have  $\bullet d = \{c_2\}$ ,  $d \bullet = \{c_4\}$  and  $\lambda(d) =$  “check ticket”. The Petri net does not contain any silent transition.

The execution semantics of Petri nets are based on the concept of *markings*. A marking  $M$  is a bag of tokens, i.e.  $M \in \mathcal{B}(P)$ . Graphically, a place  $p$ ’s marking is visualized

by drawing  $M(p)$  number of dots inside place  $p$ , e.g. place “start” in Fig. 1. A *marked Petri net* is a 2-tuple  $(N, M)$ , where  $M$  represents  $N$ ’s marking. We let  $M_i$  denote  $N$ ’s *initial marking*. Transition  $t \in T$  is *enabled* in marking  $M$  if  $\forall p \in \bullet t (M(p) > 0)$ . Enabled transition  $t$  in marking  $M$ , may *fire*, which results in new marking  $M'$ . If  $t$  fires, denoted as  $(N, M) \xrightarrow{t} (N, M')$ , then for each  $p \in P$  we have  $M'(p) = M(p) - 1$  if  $p \in \bullet t \setminus t \bullet$ ,  $M'(p) = M(p) + 1$  if  $p \in t \bullet \setminus \bullet t$ , and,  $M'(p) = M(p)$  otherwise, e.g. in Fig. 1 we have  $(W_1, [start]) \xrightarrow{a} (W_1, [c_1, c_2])$ . Given sequence  $\sigma = \langle t_1, t_2, \dots, t_n \rangle \in T^*$ ,  $\sigma$  is a *firing sequence* of  $(N, M)$ , written as  $(N, M) \xrightarrow{\sigma} (N, M')$  if and only if for  $n = |\sigma|$  there exist markings  $M_1, M_2, \dots, M_{n-1}$  such that  $(N, M) \xrightarrow{t_1} (N, M_1), (N, M_1) \xrightarrow{t_2} (N, M_2), \dots, (N, M_{n-1}) \xrightarrow{t_n} (N, M')$ . We write  $(N, M) \xrightarrow{\sigma} *$  if there exists a marking  $M'$  s.t.  $(N, M) \xrightarrow{\sigma} (N, M')$ . We write  $(N, M) \rightsquigarrow (N, M')$  if there exists  $\sigma \in T^*$  s.t.  $(N, M) \xrightarrow{\sigma} (N, M')$ .

WF-nets extend Petri nets and require the existence of a unique *source-* and *sink place* which describe the start, respectively end, of a case. Moreover, each element within the WF-net needs to be on a path from the source to the sink place.

**Definition 1** (*Workflow net* [27]) Let  $N = (P, T, F, \lambda)$  be a Petri net. Let  $p_i, p_o \in P$  with  $p_i \neq p_o$ . Let  $\Lambda \subset \mathcal{A}$  be a set of activities, let  $\tau \notin \Lambda$  and let  $\lambda: T \rightarrow \Lambda \cup \{\tau\}$ . Tuple  $W = (P, T, F, p_i, p_o, \lambda)$  is a workflow net (WF-net) if and only if:

1.  $\bullet p_i = \emptyset$
2.  $p_o \bullet = \emptyset$
3. Each element  $x \in P \cup T$  is on a path from  $p_i$  to  $p_o$ .

The execution semantics defined for Petri nets can directly be applied on the elements  $P, T$  and  $F$  of  $W = (P, T, F, p_i, p_o, \lambda)$ . Notation-wise we substitute  $W$  for its underlying net structure  $N = (P, T, F)$ , e.g.  $(W, M) \rightsquigarrow (W, M')$ . In context of WF-nets, we assume  $M_i = [p_i]$  and  $M_f = [p_o]$  unless mentioned otherwise.

Several behavioural quality metrics, that do not need any form of domain knowledge, exist for WF-nets. Several notions of *soundness* of WF-nets are defined [32]. For example, *classical sound* WF-nets are guaranteed to be free of livelocks, deadlocks, and other anomalies that can be detected automatically. In this paper we consider the weaker notion of *relaxed soundness*. Relaxed soundness requires that each transition is at some point enabled, and, after firing such transition we are able to eventually reach the final marking.

**Definition 2** (*Relaxed soundness* [32]) Let  $W = (P, T, F, p_i, p_o, \lambda)$  be a WF-net.  $W$  is relaxed sound if and only if:  $\forall t \in T (\exists M, M' \in \mathcal{B}(P) ((W, [p_i]) \rightsquigarrow (W, M) \wedge (W, M) \xrightarrow{t} (W, M') \wedge (W, M') \rightsquigarrow (W, [p_o])))$ .

Reconsider  $W_1$  (Fig. 1) and assume we are given an event log with one trace:  $\langle a, b, d, e, h \rangle$ . It is quite easy to see that  $W_1$  is relaxed sound. Moreover, replay-fitness is perfect, i.e.  $\langle a, b, d, e, h \rangle$  is in the WF-net’s labelled execution language. Precision is not perfect as the WF-net can produce a lot more traces than just  $\langle a, b, d, e, h \rangle$ .

### 3.3 Discovering petri net places using integer linear programming

In [34] an integer linear programming (ILP)-formulation [24] is presented which allows for finding places of a Petri net. A solution to the ILP-formulation corresponds to a *region*, which in turn corresponds to a Petri net place. The premise of a region is the fact that its corresponding place, given the prefix-closure of an event log, does not block the execution of any sequence within the prefix-closure. We represent a region as an assignment of binary decision variables describing the incoming and outgoing arcs of its corresponding place, as well as its marking.

Given an event log  $L$  over set of activities  $A_L$ , a region is a triple  $r = (m, \mathbf{x}, \mathbf{y})$ , with  $m \in \{0, 1\}$  and  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^{|A_L|}$ , that adheres to:

$$\forall \sigma = \sigma' \cdot \langle a \rangle \in \bar{L} (m + \mathbf{p}(\sigma')^\top \mathbf{x} - \mathbf{p}(\sigma)^\top \mathbf{y} \geq 0) \tag{3.1}$$

A region  $r$  is translated to a Petri net place  $p$  as follows. Given a Petri net structure that has a unique transition  $t_a \in T$  with  $\lambda(t_a) = a$ . If,  $\mathbf{x}(a) = 1$ , we add  $t_a$  to  $\bullet p$ . Symmetrically, if  $\mathbf{y}(a) = 1$ , we add  $t_a$  to  $p \bullet$ . Finally, if  $m = 1$ , place  $p$  is initially marked. Since translating a region to a place is deterministic, we are also able to translate a place to a region, e.g. place  $c_2$  in Fig. 1 corresponds to a region with  $\mathbf{x}(a) = 1, \mathbf{x}(f) = 1, \mathbf{y}(d) = 1$  and all other variables set to zero.

Prior to presenting the basic ILP-formulation for finding regions, we formulate regions in terms of matrices, which we use in the ILP-formulation.

**Definition 3** (*Region (matrix form)*) Given an event log  $L$  over a set of activities  $A_L$ , let  $m \in \{0, 1\}$  and let  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^{|A_L|}$ . Let  $\mathbf{M}$  and  $\mathbf{M}'$  be two  $|\bar{L} \setminus \{\epsilon\}| \times |A_L|$  matrices with  $\mathbf{M}(\sigma, a) = \mathbf{p}(\sigma)(a)$  and  $\mathbf{M}'(\sigma, a) = \mathbf{p}(\sigma')(a)$  (where  $\sigma = \sigma' \cdot \langle a' \rangle \in \bar{L}$ ). Tuple  $r = (m, \mathbf{x}, \mathbf{y})$  is a region if and only if:

$$m\mathbf{1} + \mathbf{M}'\mathbf{x} - \mathbf{M}\mathbf{y} \geq \mathbf{0} \tag{3.2}$$

We additionally define matrix  $\mathbf{M}_L$  which is an  $|L| \times |A_L|$  matrix with  $\mathbf{M}_L(\sigma, a) = \mathbf{p}(\sigma)(a)$  for  $\sigma \in L$ , i.e.,  $\mathbf{M}_L$  is the equivalent of  $\mathbf{M}$  for all traces in the event log. We define a general process discovery ILP-formulation that guarantees to find a non-trivial region, i.e. regions unequal to  $(0, \mathbf{0}, \mathbf{0})$  and  $(1, \mathbf{1}, \mathbf{1})$ , with the property that its corresponding place is always empty after replaying each trace within the event log.

**Definition 4** (*Process discovery ILP-formulation* [34]) Given an event log  $L$  over a set of activities  $A_L$  and corresponding matrices  $\mathbf{M}, \mathbf{M}'$  and  $\mathbf{M}_L$ . Let  $c_m \in \mathbb{R}$  and  $\mathbf{c}_x, \mathbf{c}_y \in \mathbb{R}^{|A_L|}$ . The process discovery ILP-formulation,  $ILP_L$ , is defined as:

<b>minimize</b>	$z = c_m m + \mathbf{c}_x^\top \mathbf{x} + \mathbf{c}_y^\top \mathbf{y}$	objective function
<b>suchthat</b>	$m\mathbf{1} + \mathbf{M}'\mathbf{x} - \mathbf{M}\mathbf{y} \geq \mathbf{0}$	theory of regions
<b>and</b>	$m\mathbf{1} + \mathbf{M}_L(\mathbf{x} - \mathbf{y}) = \mathbf{0}$	corresp. place is empty after each trace
	$\mathbf{1}^\top \mathbf{x} + \mathbf{1}^\top \mathbf{y} \geq 1$	at least one arc connected
	$\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}$	i.e. $\mathbf{x} \in \{0, 1\}^{ A }$
	$\mathbf{0} \leq \mathbf{y} \leq \mathbf{1}$	i.e. $\mathbf{y} \in \{0, 1\}^{ A }$
	$0 \leq m \leq 1$	i.e. $m \in \{0, 1\}$



Definition 4 allows us to find a region that minimizes objective function  $z = c_m m + \mathbf{c}_x^T \mathbf{x} + \mathbf{c}_y^T \mathbf{y}$ . Multiple instantiations of  $z$ , i.e. in terms of *objective coefficients*  $c_m$ ,  $\mathbf{c}_x$  and  $\mathbf{c}_y$ , are possible. In [34] an objective function is proposed that minimizes 1-values in  $\mathbf{x}$  and maximizes 1-values in  $\mathbf{y}$ , i.e. in the region's corresponding place the number of incoming arcs is minimized whereas the number of outgoing arcs is maximized. In [37] the aforementioned objective function is extended such that it minimizes the time a token resides in the corresponding place. Both objective functions are expressible as a more general function which favours *minimal regions* [37], i.e. regions that are not expressible as a non-negative linear combination of two other regions. This is interesting since non-minimal regions correspond to implicit places [34]. In this paper we simply assume that one uses such an objective function.

## 4 Discovering relaxed sound workflow nets

Using the basic formulation with some objective function instantiation only yields one, optimal, result. However, we are interested in finding multiple places that together form a workflow net. In [34] multiple approaches are presented to find multiple, different Petri net places. Here we adopt, and generalize, the *causal approach*.

### 4.1 Discovering multiple places based on causal relations

One of the most suitable techniques to find multiple regions in a controlled, structured manner, is by exploiting causal relations present within an event log. A causal relation between activities  $a$  and  $b$  implies that activity  $a$  causes  $b$ , i.e.  $b$  is likely to follow (somewhere) after activity  $a$ . Several approaches exist to compute causalities [35]. For example, in [30] a causal relation  $a \rightarrow_L b$  from activity  $a$  to activity  $b$  is defined to hold if, within some event log  $L$ , we find traces of the form  $\langle \dots, a, b, \dots \rangle$  though we do not find traces of the form  $\langle \dots, b, a, \dots \rangle$ . In [40,41] this relation was further developed to take frequencies into account as well. Given these multiple definitions, we assume the existence of a *causal relation oracle* which, given an event log, produces a set of pairs  $(a, b)$  indicating that activity  $a$  has a causal relation with (to) activity  $b$ .

**Definition 5** (*Causal relation oracle*) A causal relation oracle  $\gamma_c$  maps a bag of traces to a set of activity pairs, i.e.  $\gamma_c : \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{P}(\mathcal{A} \times \mathcal{A})$ .

A causal oracle only considers activities present in an event log, i.e.  $\gamma_c(L) \in \mathcal{P}(A_L \times A_L)$ . It defines a directed graph with  $A_L$  as vertices and each pair  $(a, b) \in \gamma_c(L)$  as an arc between  $a$  and  $b$ . Later we exploit the graph-based view, for now we refer to  $\gamma_c(L)$  as a collection of pairs. When adopting a causal ILP process discovery strategy, we try to find net places that represent a causality found in the event log. Given an event log  $L$ , for each pair  $(a, b) \in \gamma_c(L)$  we enrich the constraint body with three constraints: (1)  $m = 0$ , (2)  $\mathbf{x}(a) = 1$  and (3)  $\mathbf{y}(b) = 1$ . The three constraints ensure that if we find a solution to the ILP, it corresponds to a place which is not marked and connects transition  $a$  to transition  $b$ . Given pair  $(a, b) \in \gamma_c(L)$  we denote the corresponding extended causality based ILP-formulation as  $ILP_{(L,a \rightarrow b)}$ .

After solving  $ILP_{(L,a \rightarrow b)}$  for each  $(a, b) \in \gamma_c(L)$ , we end up with a set of regions that we are able to transform into places in a resulting Petri net. Since we enforce  $m = 0$  for each causality, none of these places is initially marked. Moreover, due to constraints based on  $m\mathbf{1} + \mathbf{M}_L(\mathbf{x} - \mathbf{y}) = \mathbf{0}$ , the resulting place is empty after replaying each trace in the input event log within the net. Since we additionally enforce  $\mathbf{x}(a) = 1$  and  $\mathbf{y}(b) = 1$ , if we find a solution to the ILP, the corresponding place has both input and output arcs and is not eligible for being a source/sink place. Hence, the approach as-is does not allow us to find WF-nets. In the next section we show that a simple pre-processing step performed on the event log, together with specific instances of  $\gamma_c(L)$ , allows us to discover WF-nets which are relaxed sound.

## 4.2 Discovering workflow nets

Consider example event log  $L_1 = [\langle a, b, d, e, g \rangle^{10}, \langle a, c, d, e, f, d, b, e, g \rangle^{12}, \langle a, d, c, e, h \rangle^9, \langle a, b, d, e, f, c, d, e, g \rangle^{11}, \langle a, d, c, e, f, b, d, e, h \rangle^{13}]$ . Observe that for each trace  $\sigma$  in  $L_1$  we have  $(W_1, [start]) \xrightarrow{\sigma} (W_1, [end])$ . Let  $A_f \subseteq A_L$  denote the set of *final activities*, i.e. activities  $a_f$  s.t. there exists a trace of the form  $\langle \dots, a_f \rangle$  in the event log. For example, for  $L_1$ ,  $A_f = \{g, h\}$ . After solving each  $ILP_{L,a \rightarrow b}$  instance based on  $\gamma_c(L)$  and adding corresponding places, we know that when we exactly replay any trace from  $L_1$ , after firing  $g$  or  $h$ , the net is empty. Since  $g$  and  $h$  never co-occur in a trace, it is trivial to add a sink place  $p_o$ , s.t. after replaying each trace in  $L_1$ ,  $p_o$  is the only place marked, i.e.  $\bullet p_o = \{f, g\}$  and  $p_o \bullet = \emptyset$  (place “end” in Fig. 1). In general, such decision is not trivial. However, a trivial case for adding a sink  $p_o$  is the case when there is only one end activity that uniquely occurs once, at the end of each trace, i.e.  $A_f = \{a_f\}$  and there exists no trace of the form  $\langle \dots, a_f, \dots, a_f \rangle$ . In such case we have  $\bullet p_o = \{a_f\}$ ,  $p_o \bullet = \emptyset$ .

A similar rationale holds for adding a source place. We define a set  $A_s$  that denotes the set of *start activities*, i.e. activities  $a_s$  s.t. there exists a trace of the form  $\langle a_s, \dots \rangle$  in the event log. For each activity  $a_s$  in  $A_s$  we know that for some traces in the event log, these are the first ones to be executed. Thus, we know that the source place  $p_i$  must connect, in some way, to the elements of  $A_s$ . Like in the case of final transitions, creating a source place is trivial when  $A_s = \{a_s\}$  and there exists no trace of the form  $\langle a_s, \dots, a_s, \dots \rangle$ , i.e. the start activity uniquely occurs once at the beginning of each trace. In such case we create place  $p_i$  with  $\bullet p_i = \emptyset$ ,  $p_i \bullet = \{a_s\}$ .

In order to be able to find a source and a sink place, it suffices to guarantee that sets  $A_s$  and  $A_f$  are of size one and their elements always occur uniquely at the start, respectively, end of a trace. We formalize this idea through the notion of *unique start/end event logs*, after which we show that transforming an arbitrary event log to such unique start/end event log is trivial.

**Definition 6** (*Unique start/end event log*) Let  $L$  be an event log over a set of activities  $A_L$ .  $L$  is a **Unique Start/End event Log** (USE-Log) if there exist  $a_s, a_f \in A_L$  s.t.  $a_s \neq a_f, \forall \sigma \in L(\sigma(1) = a_s \wedge \forall i \in \{2, 3, \dots, |\sigma|\}(\sigma(i) \neq a_s))$  and  $\forall \sigma \in L(\sigma(|\sigma|) = a_f \wedge \forall i \in \{1, 2, \dots, |\sigma| - 1\}(\sigma(i) \neq a_f))$ .

Since the set of activities  $A_L$  is finite, it is trivial to transform any event log to a USE-log. Assume we have an event log  $L$  over  $A_L$  that is not a USE-log. We generate two “fresh” activities  $a_s, a_f \in \mathcal{A}$  s.t.  $a_s, a_f \notin A_L$  and create a new event log  $L'$  over  $A_L \cup \{a_s, a_f\}$ , by adding  $\langle a_s \rangle \cdot \sigma \cdot \langle a_f \rangle$  to  $L'$  for each  $\sigma \in L$ . We let  $\pi : \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{B}(\mathcal{A}^*)$  denote such USE-transformation. We omit  $a_s$  and  $a_f$  from the domain of  $\pi$  and assume that given some USE-transformation the two symbols are known.

Clearly, after applying a USE-transformation, finding a unique source and sink place is trivial. It also provides an additional advantage considering the ability to find WF-nets. In fact, an ILP instance  $ILP_{(L,a \rightarrow b)}$  always has a solution if  $L$  is a USE-log. We provide a proof of this property in Lemma 1, after which we present an algorithm that, given specific instantiations of  $\gamma_c$ , discovers WF-nets.

**Lemma 1** (A USE-Log based causality has a solution) *Let  $L$  be an event log over a set of activities  $A_L$ . Let  $\pi : \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{B}(\mathcal{A}^*)$  denote a USE-transformation function and let  $a_s, a_f$  denote the start and end activities. For every  $(a, b) \in \gamma_c(\pi(L))$  with  $a \neq a_f$  and  $b \neq a_s$ ,  $ILP_{(\pi(L),a \rightarrow b)}$  has a solution.*

*Proof* See [38]. □

In Algorithm 1 we present an ILP-Based process discovery approach that uses a USE-log internally in order to find multiple Petri net places. For every  $(a, b) \in \gamma_c(\pi(L))$  with  $a \neq a_f$  and  $b \neq a_s$  it solves  $ILP_{(\pi(L),a \rightarrow b)}$ . Moreover, it finds a unique source and sink place.

The algorithm constructs an initially empty Petri net  $N = (P, T, F)$ . Subsequently for each  $a \in A_L \cup \{a_s, a_f\}$  a transition  $t_a$  is added to  $T$ . For each causal pair in the USE-variant of input event log  $L$ , a place  $p_{(a,b)}$  is discovered by solving  $ILP_{(\pi(L),a \rightarrow b)}$  after which  $P$  and  $F$  are updated accordingly. The algorithm adds an initial place  $p_i$  and connects it to  $t_{a_s}$  and similarly creates sink place  $p_o$  which is connected to  $t_{a_f}$ . For transition  $t_a$  related to  $a \in A_L$ , we have  $\lambda(t_a) = a$ , whereas  $\lambda(t_{a_s}) = \lambda(t_{a_f}) = \tau$ .

The algorithm is guaranteed to always find a solution to  $ILP_{(\pi(L),a \rightarrow b)}$ , hence for each causal relation a place is found. Additionally, a unique source and sink place are constructed. However, the algorithm does not guarantee that we find a connected component, i.e. requirement 3 of Definition 1. In fact, the nature of  $\gamma_c$  determines whether or not we discover a WF-net. In Theorem 1 we characterize this nature and prove, by exploiting Lemma 1, that we are able to discover WF-nets.

**Theorem 1** (There exist sufficient conditions for finding WF-nets) *Let  $L$  be an event log over a set of activities  $A_L$ . Let  $\pi : \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{B}(\mathcal{A}^*)$  denote a USE-transformation function. Let  $a_s, a_f$  denote the unique start- and end activity of  $\pi(L)$ . Let  $\gamma_c : \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{P}(\mathcal{A} \times \mathcal{A})$  be a causal oracle and consider  $\gamma_c(\pi(L))$  as a directed graph. If each  $a \in A_L$  is on a path from  $a_s$  to  $a_f$  in  $\gamma_c(\pi(L))$ , and there is no path from  $a_s$  to itself, nor a path from  $a_f$  to itself, then  $ILP\text{-Based Process Discovery}(L, \gamma_c)$  returns a WF-net.*

*Proof* See [38]. □

Theorem 1 proves that if we use a causal structure that, when interpreting it as a graph, has the property that each  $a \in A_L$  is on a path from  $a_s$  to  $a_f$ , the result

**Algorithm 1:** ILP-Based Process Discovery

---

```

input :  $L \in \mathcal{B}(A_L^*)$ ,  $\gamma_c: \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{P}(\mathcal{A} \times \mathcal{A})$ 
output:  $W = (P, T, F, p_i, p_o, \lambda)$ 
begin
1   $P, T, F \leftarrow \emptyset$ ;
2  let  $a_s, a_f \notin A_L$ ;
3   $T \leftarrow \{t_a \mid a \in A_L \cup \{a_s, a_f\}\}$ ;
4  foreach  $(a, b) \in \gamma_c(\pi(L))$  do
5    if  $a \neq a_f \wedge b \neq a_s$  then
6       $(m, \mathbf{x}, \mathbf{y}) \leftarrow$  solution to  $ILP_{(\pi(L), a \rightarrow b)}$ ;
7      let  $p_{(a,b)} \notin P$ ;
8       $P \leftarrow P \cup p_{(a,b)}$ ;
9      foreach  $a' \in A_L \cup \{a_s, a_f\}$  do
10     if  $\mathbf{x}(a') = 1$  then
11        $F \leftarrow F \cup \{t_{a'}, p_{(a,b)}\}$ ;
12     if  $\mathbf{y}(a') = 1$  then
13        $F \leftarrow F \cup \{p_{(a,b)}, t_{a'}\}$ ;
14  let  $p_i, p_o \notin P$ ;
15   $P \leftarrow P \cup \{p_i, p_o\}$ ;
16   $F \leftarrow F \cup \{p_i, t_{a_s}\}$ ;
17   $F \leftarrow F \cup \{t_{a_f}, p_o\}$ ;
18  let  $\lambda: T \rightarrow A \cup \{\tau\}$ ;
19  foreach  $a \in A_L$  do
20     $\lambda(t_a) \leftarrow a$ ;
21   $\lambda(t_{a_s}), \lambda(t_{a_f}) \leftarrow \tau$ ;
22  return  $(P, T, F, p_i, p_o, \lambda)$ ;
```

---

of ILP-Based Process Discovery( $L, \gamma_c$ ) is a WF-net. Although this seems a rather strict property of the causal structure, there exists a specific causal graph definition that guarantees this property [40]. Hence we are able to use this definition as an instantiation for  $\gamma_c$ .

Theorem 1 does not provide any behavioural guarantees, i.e. a WF-net is a purely graph-theoretical property. Recall that the premise of a region is that it does not block the execution of any sequence within the prefix-closure of an event log. Intuitively we deduce that we are therefore able to fire each transition in the WF-net at least once. Moreover, since we know that  $a_f$  is the final transition of each sequence in  $\pi(L)$ , and after firing the transition each place based on any  $ILP_{(\pi(L), a \rightarrow b)}$  is empty, we know that we are able to mark  $p_o$ . These two observations hint on the fact that the WF-net is *relaxed sound*, which we prove in Theorem 2

**Theorem 2** *Let  $L$  be an event log over a set of activities  $A_L$ . Let  $\pi: \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{B}(\mathcal{A}^*)$  denote a USE-transformation function and let  $a_s, a_f$  denote the unique start- and end activity of  $\pi(L)$ . Let  $\gamma_c: \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{P}(\mathcal{A} \times \mathcal{A})$  be a causal oracle. Let  $W = (P, T, F, p_i, p_o, \lambda) = \text{ILP-Based Process Discovery}(A, L, \gamma_c)$ . If  $W$  is a WF-net, then  $W$  is relaxed sound.*

*Proof* See [38]. □

We have shown that with a few pre- and post-processing steps and a specific class of causal structures we are able to guarantee to find WF-nets that are relaxed sound.

These results are interesting since several process mining techniques require WF-nets as an input. The ILP problems solved still require their solutions to allow for all possible behaviour in the event log. As a result, the algorithm incorporates all infrequent exceptional behaviour and still results in over-fitting complex WF-nets. Hence, in the upcoming section we show how to efficiently prune the ILP constraint body to identify and eliminate infrequent exceptional behaviour.

## 5 Dealing with infrequent behaviour

In this section we present an efficient pruning technique that identifies and eliminates constraints related to infrequent exceptional behaviour. We first present the impact of infrequent exceptional behaviour after which we present the pruning technique.

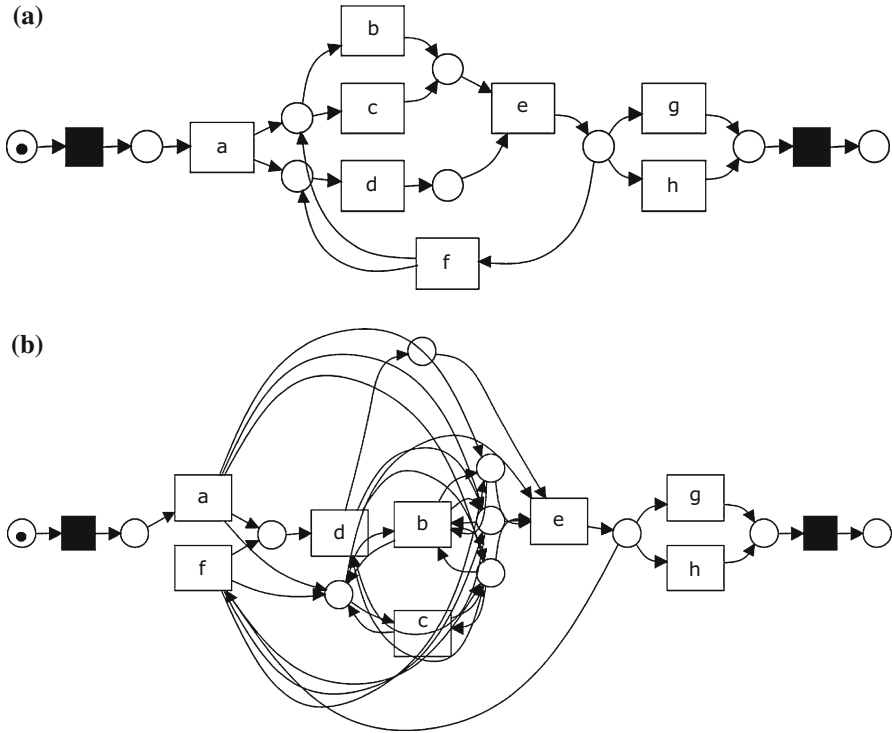
### 5.1 The impact of infrequent exceptional behaviour

In this section we highlight the main cause of ILP-based discovery’s inability to handle infrequent behaviour and we devise a filtering mechanism that exploits the nature of the underlying body of constraints.

Let us again consider example event log  $L_1$ , i.e.,  $L_1 = [\langle a, b, d, e, g \rangle^{10}, \langle a, c, d, e, f, d, b, e, g \rangle^{12}, \langle a, d, c, e, h \rangle^9, \langle a, b, d, e, f, c, d, e, g \rangle^{11}, \langle a, d, c, e, f, b, d, e, h \rangle^{13}]$ . Using an implementation of Algorithm 1 in ProM [39], with a suitable causal structure  $\gamma_c$ , we find the WF-net depicted in Fig. 2a. The WF-net describes the same behaviour as the model presented in Fig. 1 and has perfect replay-fitness w.r.t.  $L_1$ . However, if we create event log  $L'_1$  by simply adding one instance of the trace  $\langle a, b, c, d, e, g \rangle$ , we obtain the result depicted in Fig. 2b. Due to one exceptional trace, the model allows us, after executing  $a$  or  $f$ , to execute an arbitrary number of  $b$ - and  $c$ -labelled transitions. This is undesirable since precision of the resulting process model drops significantly. Thus, the addition of one exceptional trace results in a less comprehensible WF-net and reduces the precision of the resulting WF-net.

When analysing the two models we observe that they share some equal places, e.g. both models have a place  $p_{(\{a, f\}, \{d\})}$  with  $\bullet p_{(\{a, f\}, \{d\})} = \{a, f\}$  and  $p_{(\{a, f\}, \{d\})} \bullet = \{d\}$ . However, the two places  $p_{(\{a, f\}, \{b, c\})}$  with  $\bullet p_{(\{a, f\}, \{b, c\})} = \{a, f\}$  and  $p_{(\{a, f\}, \{b, c\})} \bullet = \{b, c\}$  and  $p_{(\{b, c\}, \{e\})}$  with  $\bullet p_{(\{b, c\}, \{e\})} = \{b, c\}$  and  $p_{(\{b, c\}, \{e\})} \bullet = \{e\}$  in Fig. 2a, are not present in Fig. 2b. These are “replaced” by the less desirable places containing self-loops in Fig. 2b. This is caused by the fact that the constraint body of the ILP’s based on event log  $L'_1$  contain all constraints present in the ones related to  $L_1$ , combined with the additional constraints depicted in Table 2.

For place  $p_{(\{a, f\}, \{b, c\})}$  in Fig. 2a we define a corresponding tuple  $r = (m, \mathbf{x}, \mathbf{y})$  with  $\mathbf{x}(a) = 1, \mathbf{x}(f) = 1, \mathbf{y}(b) = 1$  and  $\mathbf{y}(c) = 1$  (all other variables 0). The additional constraints in Table 2 all evaluate to  $-1$  for  $r$ , e.g. constraint  $m + \mathbf{x}(a_s) + \mathbf{x}(a) + \mathbf{x}(b) - \mathbf{y}(a_s) - \mathbf{y}(a) - \mathbf{y}(b) - \mathbf{y}(c)$  evaluates to  $0 + 0 + 1 + 0 - 0 - 0 - 1 - 1 = -1$ . In case of place  $p_{(\{b, c\}, \{e\})}$  we observe that the corresponding tuple  $r = (m, \mathbf{x}, \mathbf{y})$  with  $\mathbf{x}(b) = 1, \mathbf{x}(c) = 1$  and  $\mathbf{y}(e) = 1$ , yields a value of 1 for all constraints generated by trace  $\langle a, b, c, d, e, g \rangle$ . For the constraints having a “ $\geq 0$  right hand side” this is valid, however, for constraint  $m + \mathbf{x}(a_s) + \mathbf{x}(a) + \mathbf{x}(b) + \mathbf{x}(c) + \mathbf{x}(d) + \mathbf{x}(e) + \mathbf{x}(g) +$



**Fig. 2** Results of applying Algorithm 1 (*HybridILPMiner* package in the ProM Framework [39]) based on  $L_1$  and  $L'_1$ . **a** Result based on event log  $L_1$ . **b** Result based on event log  $L'_1$

**Table 2** Some of the newly added constraints based on trace  $\langle a, b, c, d, e, g \rangle$  in event log  $L'_1$ , starting from prefix  $\langle a, b, c \rangle$  which is not present in  $\overline{L_1}$

---


$$m + x(a_s) + x(a) + x(b) - y(a_s) - y(a) - y(b) - y(c) \geq 0$$

$$m + x(a_s) + x(a) + x(b) + x(c) - y(a_s) - y(a) - y(b) - y(c) - y(d) \geq 0$$

⋮

$$m + x(a_s) + x(a) + x(b) + x(c) + x(d) + x(e) + x(g) - y(a_s) - y(a) - y(b) - y(c) - y(d) - y(e) - y(g) - y(a_f) \geq 0$$

$$m + x(a_s) + x(a) + x(b) + x(c) + x(d) + x(e) + x(g) + x(a_f) - y(a_s) - y(a) - y(b) - y(c) - y(d) - y(e) - y(g) - y(a_f) = 0$$


---

$x(a_f) - y(a_s) - y(a) - y(b) - y(c) - y(d) - y(e) - y(g) - y(a_f) = 0$  this is not valid.

The example shows that the addition of  $\langle a, b, c, d, e, g \rangle$  yields constraints that invalidate places  $p(\{a, f\}, \{b, c\})$  and  $p(\{b, c\}, \{e\})$ . As a result the WF-net based on event log  $L'_1$  contains places with self-loops on both  $b$  and  $c$  which greatly reduces its precision and simplicity. Due to the relative infrequency of trace  $\langle a, b, c, d, e, g \rangle$  it is arguably acceptable to trade-off the perfect replay-fitness guarantee of ILP-Based

process discovery and return the WF-net of Fig. 2a, given  $L'_1$ . Hence, we need filtering techniques and/or trace clustering techniques in order to remove exceptional behaviour. However, apart from simple pre-processing, we aim at adapting the ILP-based process discovery approach itself to be able to cope with infrequent behaviour.

By manipulating the constraint body such that it no longer allows for all behaviour present in the input event log, we are able to deal with infrequent behaviour within event logs. Given the problems that arise because of the presence of exceptional traces, a natural next step is to leave out the constraints related to the problematic traces. An advantage of filtering the constraint body is the fact that the constraints are based on the prefix-closure of the event log. Thus, even if all traces are unique yet they do share prefixes, we are able to filter. Additionally, leaving out constraints decreases the size of the ILP's constraint body, which has a potential positive effect on the time needed to solve an ILP. We devise a graph-based filtering technique, i.e., *sequence encoding filtering*, that allows us to prune constraints based on trace frequency information.

### 5.2 Sequence encoding graphs

As a first step towards sequence encoding filtering we define the relationship between sequences and constraints. We do this in terms of *sequence encodings*. A sequence encoding is a vector-based representation of a sequence in terms of region theory, i.e. representing the sequence's corresponding constraint.

**Definition 7** (*Sequence encoding*) Given a set of activities  $A = \{a_1, a_2, \dots, a_n\}$ .  $\phi : A^* \rightarrow \mathbb{N}^{2|A|+1}$  denotes the sequence encoding function mapping every  $\sigma \in A^*$  to a  $2 \cdot |A| + 1$ -sized vector. We define  $\phi$  as:

$$\phi(\sigma' \cdot \langle a \rangle) = \begin{pmatrix} 1 \\ \mathbf{p}(\sigma') \\ -\mathbf{p}(\sigma' \cdot \langle a \rangle) \end{pmatrix} \phi(\epsilon) = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

As an example of a sequence encoding vector consider sequence  $\langle a_s, a, b \rangle$  originating from  $\pi(L'_1)$ , for which we have  $\phi(\langle a_s, a, b \rangle)^T = (1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0, 0, 0)$ . Sequence encoding vectors directly correspond to region theory based constraints, e.g. if we are given  $m \in \{0, 1\}$  and  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^{|A|}$  and create a vector  $\mathbf{r}$  where  $\mathbf{r}(1) = m, \mathbf{r}(2) = \mathbf{x}(a_s), \mathbf{r}(3) = \mathbf{x}(a), \dots, \mathbf{r}(10) = \mathbf{x}(h), \mathbf{r}(11) = \mathbf{x}(a_f), \mathbf{r}(12) = \mathbf{y}(a_s), \dots, \mathbf{r}(21) = \mathbf{y}(a_f)$ , then  $\phi(\langle a_s, a, b \rangle)^T \mathbf{r} = m + \mathbf{x}(a_s) + \mathbf{x}(a) - \mathbf{y}(a_s) - \mathbf{y}(a) - \mathbf{y}(b)$ . As a compact notation for  $\sigma = \sigma' \cdot \langle a \rangle$  we write  $\phi(\sigma)$  as a pair of the bag representation of the Parikh vector of  $\sigma'$  and  $a$ , i.e.  $\phi(\langle a_s, a, b \rangle)$  is written as  $([a_s, a], b)$  whereas  $\phi(\langle a_s, a, b, c \rangle)$  is written as  $([a_s, a, b], c)$ . For  $\phi(\epsilon)$  we write  $([], \perp)$ .

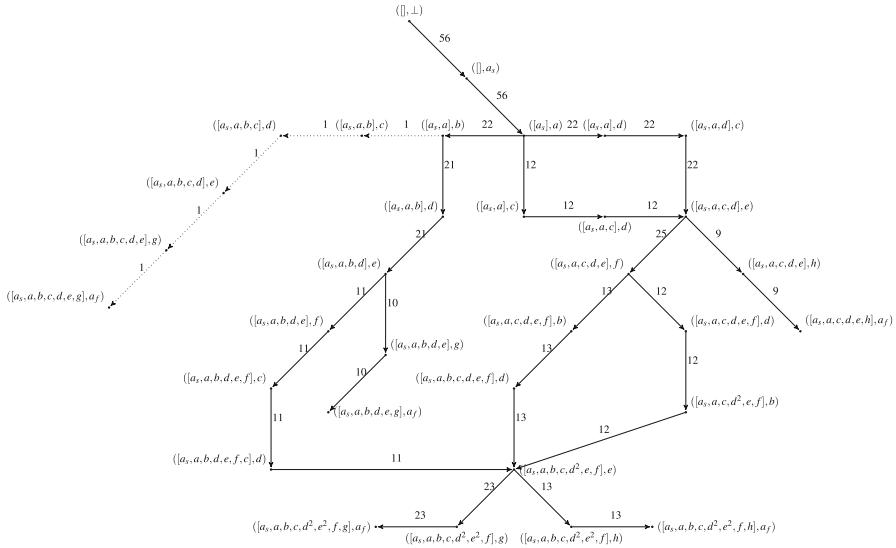
Consider the prefix-closure of  $\pi(L'_1)$  which generates the linear inequalities presented in Table 3. The table shows each sequence present in  $\pi(L'_1)$  accompanied by its  $\phi$ -value and the number of occurrences of the sequence in  $\pi(L'_1)$ , e.g.  $\overline{\pi(L'_1)}(\langle a_s, a \rangle) = 56$ . Observe that there is a relation between the occurrence of a





Table 3 continued

$\sigma \in \pi(L_1)$	$\phi(\sigma)^\top$ , i.e. $(m, \mathbf{x}(a_S), \mathbf{x}(a), \dots, \mathbf{y}(h), \mathbf{y}(a_f))$	$\phi(\sigma)$ (shorthand)	$\pi(L_1)(\sigma)$
$(a_S, a, b, d, e, f, c)$	(1, 1, 1, 0, 1, 1, 1, 0, 0, 0, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0)	$((a_S, a, b, d, e, f], c)$	11
$(a_S, a, b, d, e, g, a_f)$	(1, 1, 1, 0, 1, 1, 0, 1, 0, 0, -1, -1, -1, 0, -1, -1, 0, -1, 0, -1)	$((a_S, a, b, d, e, g], a_f)$	10
$(a_S, a, c, d, e, f, d)$	(1, 1, 1, 0, 1, 1, 1, 0, 0, 0, -1, -1, 0, -1, -2, -1, -1, 0, 0, 0)	$((a_S, a, c, d, e, f], d)$	12
$(a_S, a, d, c, e, f, b)$	(1, 1, 1, 0, 1, 1, 1, 0, 0, 0, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0)	$((a_S, a, c, d, e, f], b)$	13
$(a_S, a, d, c, e, h, a_f)$	(1, 1, 1, 0, 1, 1, 0, 0, 1, 0, -1, -1, 0, -1, -1, -1, 0, 0, -1, -1)	$((a_S, a, c, d, e, h], a_f)$	9
$(a_S, a, b, c, d, e, g, a_f)$	(1, 1, 1, 1, 1, 1, 0, 1, 0, 0, -1, -1, -1, -1, -1, -1, 0, -1, 0, -1)	$((a_S, a, b, c, d, e, g], a_f)$	1
$(a_S, a, b, d, e, f, c, d)$	(1, 1, 1, 1, 1, 1, 1, 0, 0, 0, -1, -1, -1, -1, -2, -1, -1, 0, 0, 0)	$((a_S, a, b, c, d, e, f], d)$	11
$(a_S, a, c, d, e, f, d, b)$	(1, 1, 1, 0, 1, 2, 1, 1, 0, 0, 0, -1, -1, -1, -1, -2, -1, -1, 0, 0, 0)	$((a_S, a, c, d^2, e, f], b)$	12
$(a_S, a, d, c, e, f, b, d)$	(1, 1, 1, 1, 1, 1, 1, 0, 0, 0, -1, -1, -1, -1, -2, -1, -1, 0, 0, 0)	$((a_S, a, b, c, d, e, f], d)$	13
$(a_S, a, b, d, e, f, c, d, e)$	(1, 1, 1, 1, 2, 1, 1, 0, 0, 0, -1, -1, -1, -1, -2, -2, -1, 0, 0, 0)	$((a_S, a, b, c, d^2, e, f], e)$	11
$(a_S, a, c, d, e, f, d, b, e)$	(1, 1, 1, 1, 2, 1, 1, 0, 0, 0, -1, -1, -1, -1, -2, -2, -1, 0, 0, 0)	$((a_S, a, b, c, d^2, e, f], e)$	12
$(a_S, a, d, c, e, f, b, d, e)$	(1, 1, 1, 1, 2, 1, 1, 0, 0, 0, -1, -1, -1, -1, -2, -2, -1, 0, 0, 0)	$((a_S, a, b, c, d^2, e, f], e)$	13
$(a_S, a, b, d, e, f, c, d, e, g)$	(1, 1, 1, 1, 2, 2, 1, 0, 0, 0, -1, -1, -1, -1, -2, -2, -1, -1, 0, 0)	$((a_S, a, b, c, d^2, e^2, f], g)$	11
$(a_S, a, c, d, e, f, d, b, e, g)$	(1, 1, 1, 1, 2, 2, 1, 0, 0, 0, -1, -1, -1, -1, -2, -2, -1, -1, 0, 0)	$((a_S, a, b, c, d^2, e^2, f], g)$	12
$(a_S, a, d, c, e, f, b, d, e, h)$	(1, 1, 1, 1, 2, 2, 1, 0, 0, 0, -1, -1, -1, -1, -2, -2, -1, 0, -1, 0)	$((a_S, a, b, c, d^2, e^2, f], h)$	13
$(a_S, a, b, d, e, f, c, d, e, g, a_f)$	(1, 1, 1, 1, 2, 2, 1, 1, 0, 0, -1, -1, -1, -1, -2, -2, -1, -1, 0, -1)	$((a_S, a, b, c, d^2, e^2, f, g], a_f)$	11
$(a_S, a, c, d, e, f, d, b, e, g, a_f)$	(1, 1, 1, 1, 2, 2, 1, 1, 0, 0, -1, -1, -1, -1, -2, -2, -1, -1, 0, -1)	$((a_S, a, b, c, d^2, e^2, f, g], a_f)$	12
$(a_S, a, d, c, e, f, b, d, e, h, a_f)$	(1, 1, 1, 1, 2, 2, 1, 0, 1, 0, -1, -1, -1, -1, -2, -2, -1, 0, -1, -1)	$((a_S, a, b, c, d^2, e^2, f, h], a_f)$	13



**Fig. 3** An example sequence encoding graph  $G'_1$ , based on example event log  $L'_1$

sequence and its corresponding postfixes, i.e. after the 56 times that sequence  $\langle a_s, a \rangle$  occurred,  $\langle a_s, a, b \rangle$  occurred 22 times,  $\langle a_s, a, c \rangle$  occurred 12 times and  $\langle a_s, a, d \rangle$  occurred 22 times (note:  $56 = 22 + 12 + 22$ ). Due to coupling of sequences to constraints, i.e. by means of sequence encoding, we can now apply the aforementioned reasoning to constraints as well. The frequencies in  $\pi(L'_1)$  allow us to decide whether the presence of a certain constraint is in line with predominant behaviour in the event log. For example, in Table 3,  $\phi(\langle a_s, a, b, c \rangle)$  relates to *infrequent behaviour* as it appears only once.

To apply filtering, we construct a weighted directed graph in which each sequence encoding acts as a vertex. We connect two vertices by means of an arc if the source constraint corresponds to a sequence that is a prefix of a sequence corresponding to the target constraint, i.e., we connect  $\phi(\langle a_s, a \rangle)$  to  $\phi(\langle a_s, a, b \rangle)$  as  $\langle a_s, a \rangle$  is a prefix of  $\langle a_s, a, b \rangle$ . The arc weight corresponds to trace frequency in the input event log.

**Definition 8** (*Sequence encoding graph*) Given event log  $L$  over set of activities  $A_L$ . A sequence encoding graph is a directed graph  $G = (V, E, \psi)$  where  $V = \{\phi(\sigma) \mid \sigma \in \bar{L}\}$ ,  $E \subseteq V \times V$  s.t.  $(\phi(\sigma'), \phi(\sigma)) \in E \Leftrightarrow \exists a \in A(\sigma' \cdot \langle a \rangle = \sigma)$  and  $\psi : E \rightarrow \mathbb{N}$  where:

$$\psi(v_1, v_2) = \sum_{\substack{\sigma \in \bar{L} \\ \phi(\sigma) = v_2}} \bar{L}(\sigma) - \sum_{\substack{\sigma' \in \bar{L} \\ \sigma' \cdot \langle a \rangle \in \bar{L} \\ \phi(\sigma' \cdot \langle a \rangle) = v_2 \\ \phi(\sigma') \neq v_1}} \bar{L}(\sigma')$$

Consider the sequence encoding graph in Fig. 3, based on  $\pi(L'_1)$ , as an example. By definition,  $((), \perp)$  is the root node of the graph and connects to all one-sized sequences.

Within the graph we observe the relation among different constraints, combined with their absolute frequencies based on  $L'_1$ .

### 5.3 Filtering

Given a sequence encoding graph we are able to filter out constraints. In Algorithm 2 we devise a simple breadth-first traversal algorithm, i.e. Sequence Encoding Filtering–Breadth First Search (SEF–BFS), that traverses the sequence encoding graph and concurrently constructs a set of ILP constraints. The algorithm needs a function as an input that is able to determine, given a vertex in the sequence encoding graph, what portion of adjacent vertices remains in the graph and which are removed.

**Definition 9** (*Sequence encoding filter*) Given event log  $L$  over set of activities  $A_L$  and a corresponding sequence encoding graph  $G = (V, E, \psi)$ . A sequence encoding filter is a function  $\kappa : V \rightarrow \mathcal{P}(V)$ .

Note that  $\kappa$  is an abstract function and might be parametrized. As an example consider  $\kappa_{\max}^\alpha$  which we define as:

$$\kappa_{\max}^\alpha(v) = \left\{ v' \mid (v, v') \in E \wedge \psi(v, v') \geq (1 - \alpha) \cdot \max_{v'' \in V} \psi(v, v'') \right\}, \alpha \in [0, 1]$$

Other instantiations of  $\kappa$  are possible as well and hence  $\kappa$  is a parameter of the general approach. It is however desirable that  $\kappa(v) \subseteq \{v' \mid (v, v') \in E\}$ , i.e. it only considers vertices reached by  $v$  by means of an arc. Given an instantiation of  $\kappa$ , it is straightforward to construct a filtering algorithm based on breadth-first graph traversal, i.e. SEF–BFS.

---

#### Algorithm 2: SEF-BFS

---

```

input :  $G = (V, E, \psi), \kappa : V \rightarrow \mathbb{P}(V)$ 
output:  $C \subseteq V$ 
begin
1   $C \leftarrow \emptyset$ 
2  Let  $Q$  be a FIFO queue
3   $Q.enqueue([\perp])$ 
4  while  $Q \neq \emptyset$  do
5       $v \leftarrow Q.dequeue()$ 
6      for  $v' \in \kappa(v)$  do
7           $C \leftarrow C \cup \{v'\}$ 
8           $Q.enqueue(v')$ 
9  return  $C$ 

```

---

The algorithm inherits its worst-case complexity of breadth first search, multiplied by the worst-case complexity of  $\kappa$ . Thus, in case  $\kappa$ 's worst-case complexity is  $O(1)$  then we have  $O(|V| + |E|)$  for the SEF–BFS-algorithm. It is trivial to prove, by means of induction on the length of a sequence encoding's corresponding sequence, that a sequence encoding graph is acyclic. Hence, termination is guaranteed.

As an example of executing the SEF-BFS algorithm, reconsider Fig. 3. Assume we use  $\kappa_{\max}^{0.75}$ . Vertex  $([], \perp)$  is initially present in  $Q$  and will be analysed. Since  $([], a_s)$  is the only child of  $([], \perp)$ , it is added to  $Q$ . Vertex  $([], \perp)$  is removed from the queue and is never inserted in the queue again due to the acyclic property of the graph. Similarly, since  $([a_s], a)$  is the only child of  $([], a_s)$  it is added to  $Q$ . All children of  $([a_s], a)$ , i.e.  $([a_s, a], b)$ ,  $([a_s, a], c)$  and  $([a_s, a], d)$ , are added to the queue since the maximum corresponding arc value is 22, and,  $(1 - 0.75) * 22 = 5.5$ , which is smaller than the lowest arc value 12. When analysing  $([a_s, a], b)$  we observe a maximum outgoing arc with value 21 to vertex  $([a_s, a, b], d)$  which is enqueued in  $Q$ . Since  $(1 - 0.25) * 21 = 5.25$ , the algorithm does not enqueue  $([a_s, a, b], c)$ . Note that the whole path of vertices from  $([a_s, a, b], c)$  to  $([a_s, a, b, c, d, e, g], a_f)$  is never analysed and is stripped from the constraint body, i.e. they are never inserted in  $C$ .

When applying ILP-based process discovery based on event log  $L'_1$  with sequence encoding filtering and  $\kappa_{\max}^{0.75}$ , we obtain the WF-net depicted in Fig. 2a. As explained, the filter leaves out all constraints related to vertices on the path from  $([a_s, a, b], c)$  to  $([a_s, a, b, c, d, e, g], a_f)$ . Hence, we find a similar model to the model found on event log  $L_1$  and are able to filter out infrequent exceptional behaviour.

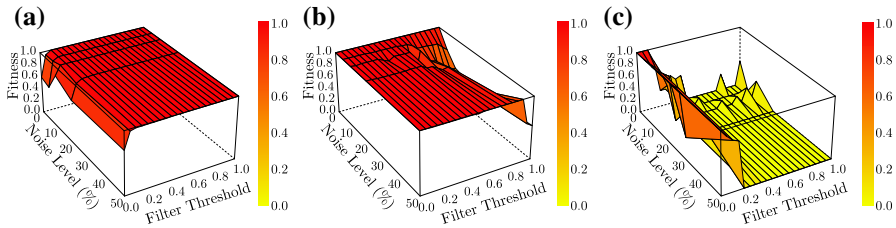
## 6 Evaluation

Algorithm 1 and sequence encoding filtering are implemented in the *HybridILP-Miner* package (<http://svn.win.tue.nl/repos/prom/Packages/HybridILPMiner/>) which is available in the *ProM* framework [39] (<http://www.promtools.org>) and the *Rapid-ProM* framework [26].<sup>1</sup> Using this implementation we validated the approach. In an artificial setting we evaluated the quality of models discovered and the efficiency of applying sequence encoding filtering. We also compare sequence encoding to the IMi [15] algorithm and automaton-based filtering [9]. Finally, we assess the performance of sequence encoding filtering on real event data [11, 19].

### 6.1 Model quality

The event logs used in the empirical evaluation of model quality are artificially generated event logs and originate from a study related to the impact of exceptional behaviour to rule-based approaches in process discovery [20]. Three event logs were generated out of three different process models, i.e. the *ground truth event logs*. These event logs do not consist of any exceptional behaviour, i.e. every trace fits the originating model. The ground truth event logs are called *a12f0n00*, *a22f0n00* and *a32f0n00*. The two digits behind the *a* character indicate the number of activities present in the event log, i.e. *a12f0n00* contains 12 different activities. From each ground truth event log, by means of trace manipulation, four other event logs are created that do contain

<sup>1</sup> Experiments are performed with source code available at: [https://github.com/rapidprom/rapidprom-source/tree/2017\\_computing\\_ilp\\_1](https://github.com/rapidprom/rapidprom-source/tree/2017_computing_ilp_1). Experiments are conducted on machines with 8 Intel Xeon CPU E5-2407 v2 @ 2.40 GHz processors and 64 GB RAM. Raw experiment results are available at: [https://github.com/s-j-v-zelst/research/releases/download/2017\\_computing/experiments.tar.gz](https://github.com/s-j-v-zelst/research/releases/download/2017_computing/experiments.tar.gz).



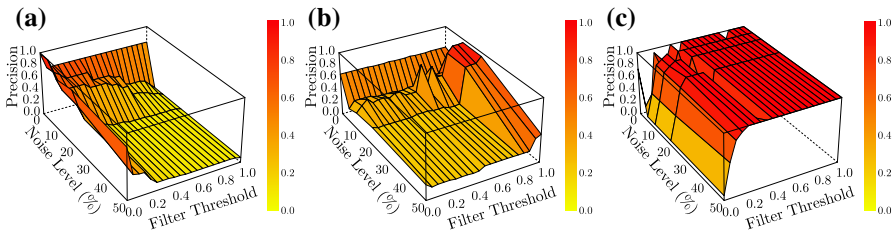
**Fig. 4** Replay-fitness measurements based on *a22f0nXX*. **a** Sequence encoding. **b** IMi [15]. **c** ILP with Automaton Filter [9]

exceptional behaviour. Manipulation concerns tail/head of trace removal, random part of the trace body removal and interchanging two randomly chosen events [20]. The percentages of trace manipulation are 5, 10, 20 and 50%. The manipulation percentage is incorporated in the last two digits of the event log’s name, i.e. the 5% manipulation version of the *a22f0n00* event log is called *a22f0n05*. In this section we only study results of the experiments using the *a22f0nXX* event logs, in [38] we report on *a12f0nXX* and *a32f0nXX* as well.

The existence of ground truth event logs, free of exceptional behaviour, is of utmost importance for evaluation. We need to be able to distinguish *normal* from *exceptional* behaviour in an *unambiguous manner*. Within evaluation, these event logs, combined with the quality dimension precision, allow us to judge how well a technique is able to filter out exceptional behaviour. Recall that precision is defined as the number of traces producible by the process model that are also present in the event log. Thus if all traces producible by a process model are present in an event log, precision is maximal, i.e. the precision value is 1. If the model allows for traces that are not present in the event log, precision is lower than 1.

If exceptional behaviour is present in an event log, the conventional ILP-based process discovery algorithm produces a WF-net that allows for all exceptional behaviour. As a result, the algorithm is unable to find any meaningful patterns within the event log. This typically leads to places with a lot of self-loops. The acceptance of exceptional behaviour by the WF-net, combined with the inability to find meaningful patterns yields a low level of precision, when using the ground truth log as a basis for precision computation. On the other hand, if we discover models using an algorithm that is more able to handle the presence of exceptional behaviour, we expect the algorithm to allow for less exceptional behaviour and find more meaningful patterns. Thus, w.r.t. the ground truth model, we expect higher precision values.

To evaluate the sequence encoding filtering approach, we have applied the ILP-based process discovery algorithm with sequence encoding filtering using  $\kappa_{max}^\alpha$  and  $\alpha = 0, 0.05, 0.1, \dots, 0.95, 1$ . Moreover, we performed similar experiments for IMi [15] (<http://svn.win.tue.nl/repos/prom/Packages/InductiveMiner/>) and automaton based event log filtering [9] (<http://svn.win.tue.nl/repos/prom/Packages/NoiseFiltering/>) combined with ILP-based discovery. We measured precision [21] and replay-fitness [33] based on the *ground truth event logs*. The replay-fitness results of the experiments are presented in Fig. 4. Precision results are presented in Fig. 5. In the charts we plot replay-fitness/precision against the noise level and filter threshold. We additionally use a colour scheme to highlight the differences in value.



**Fig. 5** Precision measurements based on *a22f0nXX*. **a** Sequence encoding. **b** IMi [15]. **c** ILP with Automaton Filter [9]

For the sequence encoding filter (Figs. 4a, 5a) we observe that replay-fitness is often 1, except for very rigorous levels of filtering, i.e.  $\alpha = 0$  and  $\alpha = 0.05$ . When applying it as rigorous as possible, i.e.  $\alpha = 0$ , we observe relatively stable replay-fitness values of around 0.6, for different levels of noise. The discovered model at 0% noise level has a precision value of 1. This implies that the filter, in the case of 0% noise, removes behaviour that is present in the ground-truth event log. Precision drops to around 0.7 for increasing levels of noise. The relative stable levels of replay-fitness and precision for increasing levels of noise when using  $\alpha = 0$  suggest that the filter only incorporates a few branches of most frequent behaviour, which is the same throughout different levels of noise. Since the precision values are lower than 1, combined with the fact that parallelism exists in the original model, it seems that the most frequent branches do incorporate some form of parallelism that generate behaviour not observed in the event log.

For the 5 and 10% noise levels we observe that threshold values in between 0 and 0.6 achieve acceptable levels of precision. These values are slightly lower than the precision values related to 0% noise, which implies that the filter in these cases is not able to remove all noise. The rapid trend towards precision values close to 0 for threshold levels above 0.6 suggests that the filter does not remove any or very little noise. For larger levels of noise we observe a steeper drop in precision. Only very low threshold levels (up to 0.2) achieve precision values around 0.3. The results suggest that these levels of noise introduce levels of variety in the data that no longer allow the sequence encoding filter to identify (in)frequent behaviour. Hence, even for low threshold values the filter still incorporates noise into the resulting process models.

For IMi (Figs. 4b, 5b) we observe similar behaviour (note that the filter threshold works inverted w.r.t. sequence encoding filtering, i.e. a value of 1 implies most rigorous filtering). However, replay-fitness drops a little earlier compared to sequence encoding filtering. The drop in precision of sequence encoding filtering is smoother than the drop in precision of IMi, i.e. there exist some spikes within the graph. Hence, the applying filtering within IMi seems to be less deterministic.

Finally, automaton based filtering (Figs. 4c, 5c) rapidly drops to replay-fitness values of 0. Upon inspection it turns out the filter returns empty event logs for the corresponding threshold and noise levels. Hence, the filter seems to be very sensitive around a threshold value in-between 0 and 0.2. The precision results for the automaton based filter (Fig. 5c) are as expected. With a low threshold value we have very low precision, except when we have a 0% noise level. Towards a threshold level of 0.2,

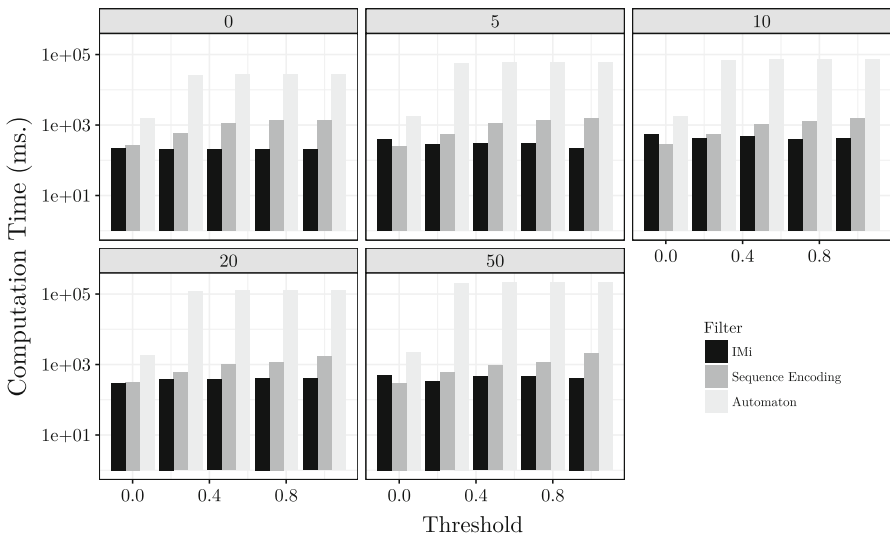
precision increases after which it maximizes out to a value of 1. This is in line with the replay-fitness measurements.

We conclude that the sequence encoding filter and IMi lead to comparable results. However, the sequence encoding filter provides more expected results, i.e. IMi behaves somewhat less deterministic. The automaton based filter does provide good results, however, sensibility of the filter threshold is much higher compared to sequence encoding filtering and IMi.

### 6.2 Computation time

Using sequence encoding filtering, we leave out constraints refer to exceptional behaviour. Hence, we reduce the size of the core ILP constraint body and thus expect a decrease in computation time when applying rigorous filtering, i.e.  $\kappa_{max}^\alpha$  with  $\alpha$  towards 0. Using RapidMiner we repeated similar experiments to the experiments performed for model quality, and measured *cpu-execution* time for the three techniques. However, we only use threshold values 0, 0.25, 0.75 and 1.

In Fig. 6 we present the average cpu-execution time, based on 50 experiment repetitions, needed to obtain a process model from an event log. For each level of noise we depict computation time for different filter threshold settings. For IMi, we measured the inductive miner algorithm with integrated filtering. For sequence encoding and automaton filtering, we measure the time needed to filter, discover a causal graph and solve underlying ILP problems. Observe that for IMi and the automaton-based filter, filtering most rigorously is performed with threshold levels of 1, as opposed to sequence encoding filtering which filters most rigorously at threshold 0.



**Fig. 6** CPU-Execution Time (ms) for a22f0nXX event logs (logarithmic scale) for different levels of noise. The percentage of noise is depicted on top of each bar chart

We observe that IMi is fastest in most cases. Computation time slightly increases when the amount of noise increases within the event logs. For sequence encoding filtering we observe that lower threshold values lead to faster computation times. This is as expected since a low threshold value removes more constraints from the ILP constraint body than a high threshold value. The automaton-based filter is slowest in all cases. The amount of noise seems to have little impact on the computation time of the automaton-based filter, it seems to be predominantly depending on the filter threshold. From Fig. 6 we conclude that IMi in general out-performs sequence encoding in terms of computation time. However, sequence encoding, in turn out-performs automaton-based filtering.

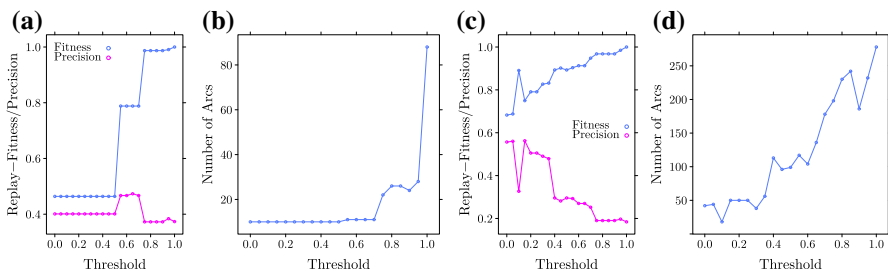
### 6.3 Application to real-life event logs

We tested the applicability of sequence encoding filtering using real-life event logs. We used an event log related to a road fines administration process [11] and one regarding the treatment of patients suspected to have sepsis [19].

The results are presented in Fig. 7. In case of the *Road Fines* event log (figures on the left-hand side of Fig. 7) we observe that replay-fitness is around 0.46 whereas precision is around 0.4 for  $\alpha$ -values from 0 to 0.5. The number of arcs for the models of these  $\alpha$ -values remains constant (as well as the number of places and the number of transitions) suggesting that the models found are the same. After this the replay-fitness increases further to around 0.8 and reaches 1 for an  $\alpha$ -level of 1. Interestingly, precision shows a little increase around  $\alpha$ -levels between 0.5 and 0.75 after which it drops slightly below its initial value. In this case, an  $\alpha$ -level in-between 0.5 and 0.75 seems most appropriate in terms of replay-fitness, precision and simplicity.

In case of the *Sepsis* event log (figures on the left-hand side of Fig. 7) we observe that replay-fitness and precision are roughly behaving as each-other's inverse, i.e. replay-fitness increases whereas precision decreases for increasing  $\alpha$ -levels. We moreover observe that the number of arcs within the process models is steadily increasing for increasing  $\alpha$ -levels. In this case, an  $\alpha$ -level in-between 0.1 and 0.4 seems most appropriate in terms of replay-fitness, precision and simplicity.

Finally, for each experiment we measured the associated computation time of solving all ILP problems. In case of the *Road Fines* event log, solving all ILP problems



**Fig. 7** **a** Fitness and precision. **b** Number of arcs. **c** Fitness and precision. **d** Number of arcs. Replay-fitness, precision and complexity based on the *Road Fines* log [11] (Fig. 7a, b) and the *Sepsis* log [19] (Fig. 7c, d)



takes roughly 5 s. In case of the `Sepsis` event log, obtaining a model ILP problems takes less than 1 s.

As our experiments show, there is no specific threshold most suitable for sequence encoding, i.e. this greatly depends on the event log. We do however observe that using lower threshold values, e.g. 0–0.4, leads to less complex models. We therefore, in practical settings, advise to use a lower threshold value first, which also reduces computation time due to a smaller constraint body size, and based on the obtained result increase or decrease the threshold value if necessary.

## 7 Conclusion

The work presented in this paper is motivated by the observation that existing region-based process discovery techniques are useful, as they are able to find non-local complex control flow patterns. However, the techniques do not provide any structural guarantees w.r.t. the resulting process models, and, they are unable to cope with infrequent, exceptional behaviour in event logs.

The approach presented in this paper extends techniques presented in [34,36,37]. We have proven that our approach is able to discover relaxed sound workflow nets, i.e. we are now able to guarantee structural properties of the resulting process model. Additionally, we presented the sequence encoding filtering technique which enables us to apply filtering exceptional behaviour within the ILP-based process discovery algorithm. Our experiments confirm that the technique enables us to find meaningful Petri net structures in data consisting of exceptional behaviour, using ILP-based process discovery as an underlying technique. Sequence encoding filtering proves to be comparable to the IMi [15] approach, i.e. an integrated filter of the Inductive Miner [16], in terms of filtering behaviour. Moreover, it is considerably faster than the general purpose filtering approach of [9] and less sensible to variations in the filter threshold.

## 8 Future work

An interesting direction for future work concerns combining ILP-based process discovery techniques with other process discovery techniques. The Inductive Miner discovers sound workflow nets, however, these models lack the ability to express complex control flow patterns such as a milestone pattern. Some of these patterns are however reconstructible using ILP-based process discovery. Hence, it is interesting to combine these approaches with possibly synergetic effects w.r.t. the process mining quality dimensions.

Another interesting approach is the development of more advanced general purpose filtering techniques. Most discovery algorithms assume the input event logs to be free of noise, infrequent and/or exceptional behaviour. Real-life event logs however typically contain a lot of such behaviour. Surprisingly, little research is performed towards filtering techniques that greatly enhance process discovery results, independent of the discovery algorithm used.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

1. Badouel E, Bernardinello L, Darondeau P (1995) Polynomial algorithms for the synthesis of bounded nets. In: Mosses PD, Nielsen M, Schwartzbach MI (eds) TAPSOFT'95: theory and practice of software development, 6th international joint conference CAAP/FASE, Aarhus, Denmark, May 22–26, 1995, Proceedings. Lecture notes in computer science, vol 915, pp 364–378. Springer, Berlin
2. Bergenthum R, Desel J, Lorenz R, Mauser S (2008) Synthesis of Petri nets from finite partial languages. *Fundam Inf* 88(4):437–468
3. Bergenthum R, Desel J, Lorenz R, Mauser S (2007) Process mining based on regions of languages. In: Alonso G, Dadam P, Rosemann M (eds) 5th international conference on business process management, BPM 2007, Brisbane, Australia, September 24–28, 2007, Proceedings. Lecture notes in computer science, 4714, pp 375–383. Springer
4. Bernardinello L (1993) Synthesis of net systems. In: Marsan MA (ed) 14th International conference on application and theory of Petri nets 1993, Chicago, IL, USA, June 21–25, 1993, Proceedings. Lecture notes in computer science, vol 691, pp 89–105. Springer
5. Bolt A, de Leoni M, van der Aalst WMP (2016) Scientific workflows for process mining: building blocks, scenarios, and implementation. *STTT* 18(6):607–628
6. Buijs JCAM, van Dongen BF, van der Aalst WMP (2012) A genetic algorithm for discovering process trees. In: Proceedings of the IEEE congress on evolutionary computation, CEC 2012, Brisbane, Australia, June 10–15, 2012, pp 1–8. IEEE
7. Buijs JCAM, van Dongen BF, van der Aalst WMP (2012) On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman R, Panetto H, Dillon TS, Rinderle-Ma S, Dadam P, Zhou X, Pearson S, Ferscha A, Bergamaschi S, Crux IF (eds) On the move to meaningful internet systems: OTM 2012, confederated international conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10–14, 2012. Proceedings, Part I. Lecture notes in computer science, vol 7565, pp 305–322. Springer
8. Carmona J, Cortadella J (2014) Process discovery algorithms using numerical abstract domains. *IEEE Trans Knowl Data Eng* 26(12):3064–3076
9. Conforti R, La Rosa M, ter Hofstede AHM (2017) Filtering out infrequent behavior from business process event logs. *IEEE Trans Knowl Data Eng* 29(2):300–314
10. Darondeau P (1998) Deriving unbounded petri nets from formal languages. In: Sangiorgi D, de Simone R (eds) 9th international conference on CONCUR '98: concurrency theory, Nice, France, September 8–11, 1998, Proceedings. Lecture notes in computer science, vol 1466, pp 533–548. Springer
11. de Leoni M, Mannhardt F (2015) Road traffic fine management process. <https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5>
12. de Weerd J, de Backer M, Vanthienen J, Baesens B (2012) A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Inf Syst* 37(7):654–676
13. Ehrenfeucht A, Rozenberg G (1990) Partial (set) 2-structures. Part I: basic notions and the representation problem. *Acta Inf* 27(4):315–342
14. Ehrenfeucht A, Rozenberg G (1990) Partial (set) 2-structures. Part II: state spaces of concurrent systems. *Acta Inf* 27(4):343–368
15. Leemans SJJ, Fahland D, van der Aalst WMP (2013) Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann N, Song M, Wohed P (eds) Business process management workshops—BPM 2013 international workshops, Beijing, China, August 26, 2013, Revised papers. *Lecture notes in business information processing*, vol 171, pp 66–78. Springer
16. Leemans SJJ, Fahland D, van der Aalst WMP (2013) Discovering block-structured process models from event logs—a constructive approach. In: Colom JM, Desel J (eds) Application and theory of Petri nets and concurrency—34th international conference, Petri Nets 2013, Milan, Italy, June 24–28, 2013. Proceedings. Lecture notes in computer science, vol 7927, pp 311–329. Springer
17. Lorenz R, Juh G (2006) Towards synthesis of Petri nets from scenarios. In: Donatelli S, Thiagarajan PS (eds) Petri nets and other models of concurrency—ICATPN 2006, 27th international conference

- on applications and theory of Petri nets and other models of concurrency, Turku, Finland, June 26–30, 2006, Proceedings. Lecture notes in computer science, vol 4024, pp 302–321. Springer
18. Lorenz R, Mauser S, Juh G (2007) How to synthesize nets from languages—a survey. In: Henderson SG, Biller B, Hsieh MH, Shortle J, Tew JD, Barton RR (eds) Proceedings of the winter simulation conference, WSC 2007, Washington, DC, USA, December 9–12, 2007, pp 637–647. WSC
  19. Mannhardt F (2016) Sepsis cases—event log. <https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>
  20. Maruster L, Weijters AJMM, van der Aalst WMP, van den Bosch A (2006) A rule-based approach for process discovery: dealing with noise and imbalance in process logs. *Data Min Knowl Discov* 13(1):67–87
  21. Munoz-Gama J (2016) Conformance checking and diagnosis in process mining—comparing observed and modeled processes. *Lecture notes in business information processing*, vol 270. Springer
  22. Murata T (1989) Petri nets: properties, analysis and applications. *Proc IEEE* 77(4):541–580
  23. Reisig W (2013) The synthesis problem. *Trans Petri Nets Other Models Concurr* 7:300–313
  24. Schrijver A (1999) Theory of linear and integer programming. Wiley-Interscience series in discrete mathematics and optimization. Wiley, London
  25. Solé M, Carmona J (2010) Process mining from a basis of state regions. In: Lilius J, Penczek W (eds) Applications and theory of Petri nets, 31st international conference, Petri nets 2010, Braga, Portugal, June 21–25, 2010, Proceedings. Lecture notes in computer science, vol 6128, pp 226–245. Springer
  26. van der Aalst WMP, Bolt A, van Zelst SJ (2017) RapidProM: mine your processes and not just your data. CoRR abs/1703.03740
  27. van der Aalst WMP (1998) The application of Petri nets to workflow management. *J Circuits Syst Comput* 8(1):21–66
  28. van der Aalst WMP (2016) Process mining—data science in action, 2nd edn. Springer, Berlin
  29. van der Aalst WMP, ter Hofstede AHM, Kiepuszewski B, Barros AP (2003) Workflow patterns. *Distrib Parallel Datab* 14(1):5–51
  30. van der Aalst WMP, Weijters AJMM, Maruster L (2004) Workflow mining: discovering process models from event logs. *IEEE Trans Knowl Data Eng* 16(9):1128–1142
  31. van der Aalst WMP, Rubin V, Verbeek HMW, van Dongen BF, Kindler E, Günther CW (2010) Process mining: a two-step approach to balance between underfitting and overfitting. *Softw Syst Model* 9(1):87–111
  32. van der Aalst WMP, van Hee KM, ter Hofstede AHM, Sidorova N, Verbeek HMW, Voorhoeve M, Wynn MT (2011) Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp Comput* 23(3):333–363
  33. van der Aalst WMP, Adriansyah A, van Dongen BF (2012) Replaying history on process models for conformance checking and performance analysis. *Wiley Interdiscipl Rew Data Min Knowl Discov* 2(2):182–192
  34. van der Werf JMEM, van Dongen BF, Hurkens CAJ, Serebrenik A (2009) Process discovery using integer linear programming. *Fundam Info* 94(3–4):387–412
  35. van Dongen BF, de Medeiros AKA, Wen L (2009) Process mining: overview and outlook of Petri net discovery algorithms. *Trans Petri Nets Other Models Concurr* 2:225–242
  36. van Zelst SJ, van Dongen BF, van der Aalst WMP (2015) Avoiding over-fitting in ILP-based process discovery. In: Motahari-Nezhad HR, Recker J, Weidlich M (eds) Business process management—13th international conference, BPM 2015, Innsbruck, Austria, August 31–September 3, 2015, Proceedings. Lecture notes in computer science, vol 9253, pp 163–171. Springer
  37. van Zelst SJ, van Dongen BF, van der Aalst WMP (2015) ILP-based process discovery using hybrid regions. In: van der Aalst WMP, Bergenthum R, Carmona J (eds) Proceedings of the ATAED 2015 workshop, satellite event of Petri Nets/ACSD 2015, Brussels, Belgium, June 22–23, 2015. CEUR workshop proceedings, vol 1371 pp 47–61. CEUR-WS.org
  38. van Zelst SJ, van Dongen BF, van der Aalst WMP, Verbeek HMW (2017) Discovering relaxed sound workflow nets using integer linear programming. CoRR abs/1703.06733
  39. Verbeek HMW, Buijs JCAM, van Dongen BF, van der Aalst WMP (2010) XES, XESame, and ProM 6. In: Soffer P, Proper E (eds) Information systems evolution—CAiSE Forum 2010, Hammamet, Tunisia, June 7–9, 2010, Selected extended papers. *Lecture notes in business information processing*, vol 72, pp 60–75. Springer

40. Weijters AJMM, Ribeiro JTS (2011) Flexible heuristics miner (FHM). In: Proceedings of the IEEE symposium on computational intelligence and data mining, CIDM 2011, part of the IEEE symposium series on computational intelligence 2011, April 11–15, 2011, Paris, France, pp 310–317
41. Weijters AJMM, van der Aalst WMP (2003) Rediscovering workflow models from event-based data using little thumb. *Integr Comput-Aided Eng* 10(2):151–162