# Filter Techniques for Region-Based Process Discovery

S.J. van Zelst, B.F. van Dongen, W.M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands
{s.j.v.zelst,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

**Abstract.** The goal of process discovery is to learn a process model based on example behavior recorded in an event log. Region-based process discovery techniques are able to uncover complex process structures (e.g., milestones) and, at the same time, provide formal guarantees w.r.t. the model discovered. For example, it is possible to ensure that the discovered model is able to replay the event log and that there are bounds on the amount of additional behavior allowed by the model that is not present in the event log. Unfortunately, region-based discovery techniques cannot handle exceptional behavior. The presence of a few exceptional traces may result in an incomprehensible model concealing the dominant behavior observed. Hence, despite their promise, region-based approaches cannot be applied in everyday process mining practice. This paper addresses the problem by proposing two filtering techniques tailored towards ILP-based process discovery (an approach based on integer linear programming and language-based region theory). Both techniques help to produce models that are less over-fitting w.r.t. the event log and have been implemented in ProM. One of the techniques is also feasible in real-life settings as it, in most cases, reduces computation time compared to conventional region-based techniques. Additionally the technique is able to produce understandable process models that better capture the dominant behavior present in the event log.

**Keywords:** Process mining, process discovery, integer linear programming, filtering

## 1 Introduction

Process mining [1] aims to assist in the improvement and understandability of business processes. Within the field, three main branches are identified. *Process discovery* aims at the construction of a process model given an event log. *Conformance checking* aims at assessing the conformance of an event log to a given process model. *Process enhancement* aims at extending, improving or repairing an existing process model using the two aforementioned disciplines as a basis. In process mining, a process model's quality is typically evaluated w.r.t. four essential quality dimensions [2]. *Replay fitness* describes to what extent the discovered model is able to reproduce the behavior present in the event log. *Precision* describes what fraction of the behavior allowed by the model is present in the event log. *Generalization* describes to what extent the model is able to reproduce future, unseen behavior of the process. *Simplicity* of a model describes the (perceived) complexity of the resulting model.

Region-based process discovery originates from the area of *Petri net synthesis* [3]. Within Petri net synthesis the main problem is deciding whether there exists a Petri net that exactly describes a given sequential behavioral system description. Two classes of region-based approaches exist: *state-based* and *language-based*. By nature, classical region-based techniques discover models that have perfect replay-fitness and very high precision. As a consequence, the models are often *extremely complex* and *severely over-fitting* w.r.t. the event log.

In the language-based region approach, a set of linear inequalities based on the event log is used as a basis. A technique to solve a body of linear inequalities is Integer Linear Programming (ILP). When applying ILP, we do not only find a feasible solution to the set of inequalities (if such solution exists) but additionally we find the optimal one, given some objective function. In [4] an algorithm was proposed that combines ILP, based on language-based region theory, with causal relations between activities present in the event log. By using causal relations, the algorithm tries to tackle the over-fitting behavior of classical language-based region techniques whilst preserving perfect replay-fitness.

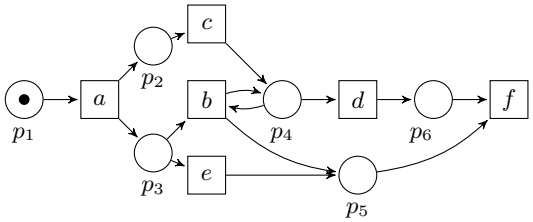
The ILP-based process discovery algorithm works well under the assumption that the event log only holds frequent behavior. Real event logs however typically also include low-frequent exceptional behavior. Such behavior may be caused by people deviating from some normative process (if this exists). Moreover, some cases may require special treatment and people may solve unexpected issues in an ad-hoc fashion. Showing all irregularities together with the "normal behavior" yields incompressible models. The presence of exceptional behavior in event logs impacts the result of applying the ILP-based process discovery algorithm significantly. As the algorithm guarantees perfect replay-fitness, it guarantees that the resulting model allows for all exceptional behavior present in the event log. In practice this leads to models that are incapable of capturing the dominant behavior present in the event log.

To leverage the strict replay-fitness guaranteed by the ILP-based process discovery algorithm we present *two filtering techniques*, tightly coupled to the algorithm's ILP formulation. One of the techniques extends the basic ILP formulation whereas the other approach exploits the underlying data abstraction used within the ILP formulation. Using simple examples we show that both approaches enable us to filter exceptional behavior from event logs and result in models that do not have perfect replay-fitness w.r.t. the input data. However, the models are simpler and less over-fitting. Moreover, the data abstraction based filtering technique is also applicable on realistically sized event logs in terms of computation time. To evaluate the technique we have applied it on a set of artificially generated event logs with varying levels of exceptional behavior.

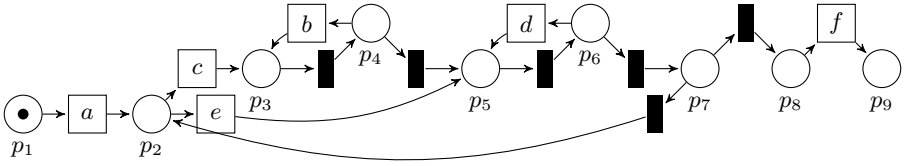
The outline of this paper is as follows. In Section 2 we motivate the need for a region-based technique able to cope with the presence of exceptional behavior. In Section 3 we present the basics of ILP-based process discovery in an informal fashion. In Section 4 we introduce two new integrated ILP filtering techniques. In Section 5 we evaluate the data abstraction based approach in terms of its effects on model quality and computation time. Section 6 concludes the paper.

## 2 Motivation

The ILP-based process discovery algorithm uses Petri nets[1] as a process model formalism. The representational bias of Petri nets, i.e. the (practical) limitations of the process model of choice, still allows for expressing complex patterns within event data, a very valuable property from a business management perspective. Many process discovery algorithms however use process model formalisms with a different representational bias, i.e. they assume models to be structured or assume that there are only local dependencies amongst activities (e.g., subclasses of free-choice nets). As an example consider the two Petri nets depicted in Figure 1, depicting the results of both the ILP-based process discovery algorithm as the Inductive Miner [5, 6] on the simple event log $L = [\langle a, c, d, e, f\rangle^{10}, \langle a, c, b, d, f\rangle^{10}, \langle a, c, e, d, f\rangle^{10}, \langle a, e, c, d, f\rangle^{10}]$[2]. The event log contains behavior that is generated by a model that exhibits a *milestone pattern* [7]. The milestone pattern enforces that transition $b$ can only fire if transition $c$ has fired. Moreover transition $b$ can only fire if transitions $e$ and/or $d$ did not fire. Finally, if transition $b$ fires, transition $e$ can no longer fire. The ILP-based discovery algorithm allows us to discover the milestone pattern whereas the inductive miner clearly neglects the pattern and results in a rather under-fitting Petri net. Other techniques such as the Heuristics miner [8], the Fuzzy miner [9] and the Genetic miner [10], like the Inductive Miner,




(a) Process discovery result of the ILP-based algorithm.




(b) Process discovery result of Inductive Miner [5, 6].

Fig. 1: Petri nets discovered using the ILP-based process discovery algorithm and the Inductive miner, based on a log that consists of behavior generated by a milestone pattern.

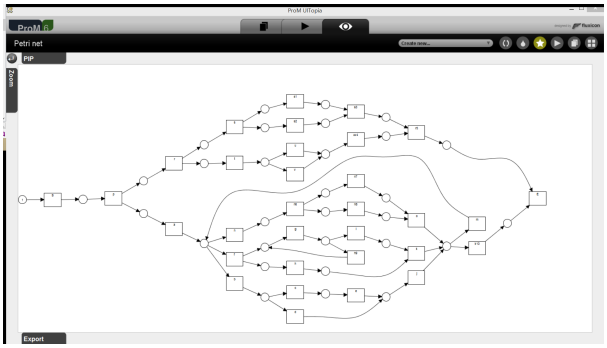
[1] In this paper we assume the reader to be generally acquainted with the concepts of event logs and Petri nets. For a detailed description of both concepts we refer to [1]. In the remainder of this paper when we use the term Petri net, we refer to a Petri net without arc weights unless explicitly mentioned otherwise.

[2] For event logs we use the notion of a mutliset of traces, using a control-flow perspective.

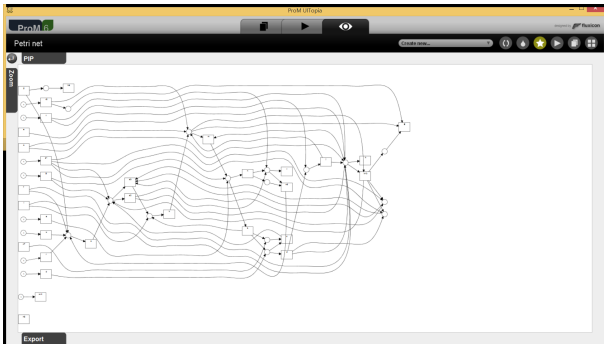
suffer from the representational bias of the process model of choice and do not use the right modeling formalism to capture this type of patterns.

The Petri nets discoverable by the ILP-based process discovery algorithm also suffer from a form of representational bias. By nature of the underlying data abstraction used, they do not allow for duplicate transition labels or the use of silent (invisible) transitions. A selection of patterns that the ILP-based process discovery algorithm is able to reproduce are patterns like *interleaved parallel routing*, *critical section* and *arbitrary cycles*. An example of patterns not supported by the ILP-based process discovery algorithm are patterns related to *OR-Split* (i.e. non-exclusive) and *Structured Synchronizing Merge* patterns [7].

The impact of exceptional behavior w.r.t. ILP-based process discovery becomes apparent when regarding the two Petri nets depicted in Figure 2. Both Petri nets are discovered using an implementation of the ILP-based process discovery algorithm in the process mining framework ProM. The event log used to discover the Petri net in Figure 2a does not contain exceptional behavior, i.e. it only consists of traces that fit




(a) Process discovery result of the ILP-based algorithm using an event log that does not contain exceptional behavior.




(b) Process discovery result of the ILP-based algorithm using an event log that contains a minimal amount of exceptional behavior.

Fig. 2: Discovered Petri nets after applying the ILP-based process discovery algorithm on event logs with and without the presence of exceptional behavior.

the Petri net presented. The event log used for discovery of the Petri net depicted in Figure 2b is a slightly manipulated version of the event log corresponding to the Petri net depicted in Figure 2a. The event log contains little exceptional behavior, i.e. $5\%$ of the traces in the event log is manipulated. Clearly, the Petri net depicted in Figure 2b is not capturing the most dominant behavior present in the event log, caused by the strict replay-fitness property guaranteed by the ILP-based process discovery algorithm. Thus, the presence of only a minimal amount of exceptional behavior can greatly influence the quality of the process models discovered by the ILP-based process discovery algorithm.

The strictness of the ILP-based process discovery algorithm enforces exceptional behavior to be part of the resulting process models. Thus, removing exceptional behavior comes as a natural next step as it enables the algorithm to discover models that more accordingly represent the behavior present in the event log. Making use of the underlying data abstraction of the ILP-miner, which is based on the *prefix-closure* of the event log, provides the ability to filter based on prefixes of traces rather than on complete traces only. In this way, filtering is applicable if all traces in an event log have equal or very similar frequencies, yet differ in terms of the occurrence of their prefixes.

## 3 ILP-Based Process Discovery

### 3.1 Language-Based Region Theory

The fundamental requirement behind language-based region theory, which forms the basis of ILP-based process discovery, is best explained by the following sentence: *Any place present in the resulting Petri net must allow each event in the input event log to be executed*.

Consider the simple event log $L = [\langle a, b, c, d, e, g\rangle^{10}, \langle a, c, b, d, e, g\rangle^{10}, \langle a, b, c, d, e, f, e, g\rangle^{10}, \langle a, c, b, d, e, f, e, g\rangle^{10}]$ and a corresponding Petri net depicted in Figure 3. Each place in the Petri net adheres to the aforementioned requirement of language-based region theory, i.e., each place allows for the execution of each event present in the event log. For example, consider place $p_5$ having an incoming arc from transition $c$ and an outgoing arc to transition $d$. As place $p_5$ is not having an outgoing arc to transitions $a, b, c, e, f$ and $g$ it does not interfere with firing these transitions at any point in time. The only outgoing arc of place $p_5$ allows for firing $d$, only after firing transition $c$. This

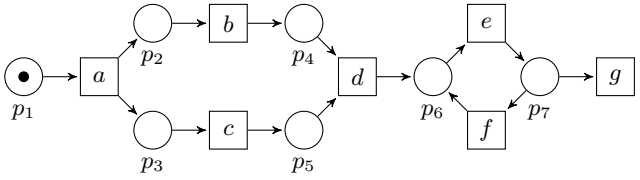



Fig. 3: A Petri net corresponding to $L = [\langle a, b, c, d, e, g\rangle^{10}, \langle a, c, b, d, e, g\rangle^{10}, \langle a, b, c, d, e, f, e, g\rangle^{10}, \langle a, c, b, d, e, f, e, g\rangle^{10}]$ and adhering to the basic concept of language-based regions.

is in line with the event log because if event $d$ occurs, it is always (indirectly) preceded by event $c$.

The impact of exceptional behavior on language-based region theory becomes apparent when we consider event log $L' = [\langle a, b, c, d, e, g\rangle^{10}, \langle a, c, b, d, e, g\rangle^{10}, \langle a, b, c, d, e, f, e, g\rangle^{10}, \langle a, c, b, d, e, f, e, g\rangle^{10}, \langle a, b, d, e, f, e, g\rangle]$. Event log $L'$ consists of one exceptional trace w.r.t. event log $L$, being $\langle a, b, d, e, f, e, g\rangle$. Given $L'$ we identify that place $p_5$ is no longer an acceptable place w.r.t. language-based region theory. This is due to the fact that place $p_5$ only allows for firing transition $d$ after transition $c$ has fired. Thus place $p_5$ does not allow for event $d$ in the newly added exceptional trace $\langle a, b, d, e, f, e, g\rangle$. If we are explicitly interested in finding a place that connects transitions $c$ and $d$ we should at least add an incoming arc to $p_5$, either from transition $a$ or from transition $b$, to make the place acceptable again w.r.t. language-based region theory. From a process discovery perspective however, the Petri net depicted in Figure 3, which is not discoverable using the conventional ILP-based process discovery algorithm when using event log $L'$ as an input, is an adequate and possibly most desired representation of the behavior captured in $L'$.

### 3.2 Applying Discovery

Given an event log it is straightforward to construct linear inequalities over a set of decision variables which express the previously described requirement of language-based regions. The linear inequalities allow us to decide whether a place may be added to the resulting Petri net. As identified in [4], the linear inequalities in turn lend themselves perfectly to act as a basis for Integer Linear Programming (ILP). Within the ILP-based approach, the linear inequalities are used as a basic body of constraints. However, the constraints only cover means to accept or reject potential places. Therefore the constraint body needs to be enriched with some objective function defined over the decision variables, which together define an ILP formulation. A multitude of possible objective functions exist, all yielding potentially different results, e.g. maximizing versus minimizing the number of arcs used by a place.

When solving an ILP, we typically end up with one optimal solution, given the objective. Therefore, the ILP-based process discovery algorithm as presented in [4] solves multiple slightly extended versions of the basic formulation in order to find specific places. The most useful approach is the usage of *causal relations* between activities exhibited within the event log. Given some algorithm that is able to learn causal relations within an event log, the ILP-based process discovery algorithm will add constraints for each causal relation and solve the corresponding extended ILPs. Each solution to a solved ILP represents a place and is added to the resulting Petri net.

The implementation related to the formulation as presented in [4], i.e. the ILPMiner[3] [11] ProM plug-in, uses one specific implementation to generate causal relations given an event log. The implementation related to the work presented in this paper, i.e. the HybridILPMiner[4] ProM plug-in, enables the user to specify what algorithm should

[3] `https://svn.win.tue.nl/repos/prom/Packages/ILPMiner/Trunk/`

[4] `https://svn.win.tue.nl/repos/prom/Packages/HybridILPMiner/Trunk/`

be used to calculate a set of causal relations. The algorithms, in turn, are implemented in the CausalActivityMatrixMiner[5] ProM plug-in. For the purpose this paper we have used the "mini" causal activity algorithm with auto-adjust to find causalities.

The causal relation based approach limits the number of places added to the resulting Petri net, i.e. it is bound by the number of causal relations exhibited in the event log. As a consequence the models found by the algorithm are less over-fitting the event log and are likely to be less precise when compared to traditional language-based region theory approaches. In fact, using different causal relations might yield different results on the same event log. As we have seen however, the technique is still extremely prone to the presence of exceptional behavior.

## 4 Integrated ILP Filtering

In order to check whether a place may be added to the resulting Petri net, we encode the *prefix-closure* of the event log as a set of linear inequalities. Reconsider event log $L = [\langle a,b,c,d,e,g\rangle^{10}, \langle a,c,b,d,e,g\rangle^{10}, \langle a,b,c,d,e,f,e,g\rangle^{10}, \langle a,c,b,d,e,f,e,g\rangle^{10}]$. $L$'s prefix closure is the set of sequences $\overline{L}$ s.t. each sequence in $\overline{L}$ is either a prefix of a trace in $L$ or a prefix of a sequence in $\overline{L}$, i.e. $\overline{L} = \{\epsilon, \langle a\rangle, \langle a,b\rangle, \langle a,c\rangle, \langle a,b,c\rangle, \langle a,c,b\rangle, ..., \langle a,b,c,d,e,f,e,g\rangle, \langle a,c,b,d,e,f,e,g\rangle\}$. As the linear inequalities do not take trace-frequencies into account we are, w.r.t. construction of constraints, not interested in the frequencies of the sequences in $\overline{L}$. In the remainder of this paper, let $A$ denote to the set of event-classes present in $L$, i.e. $A = \{a,b,c,d,e,f,g\}$.

To encode the prefix-closure we use three basic decision variables that allow us to express a place in the resulting Petri net. Variable $m \in \{0,1\}$ denotes the presence of a token, variable $\vec{x} \in \{0,1\}^{|A|}$ (an $A$-sized vector) denotes incoming arcs and variable $\vec{y} \in \{0,1\}^{|A|}$ denotes outgoing arcs. Thus as an example, for $p_1$ in Figure 3 we have $m = 1$ and $\vec{y}(a) = 1$ (all other variables are zero, i.e. $\vec{x}(a) = \vec{x}(b) = ... = \vec{y}(g) = 0$) whereas for $p_5$ we have $\vec{x}(c) = 1$ and $\vec{y}(d) = 1$.

We encode $\overline{L}$ as a minimal sequence of token production and consumption steps, using the aforementioned variables and the (flower) Petri net depicted in Figure 4 (note that for place $p$ in Figure 4 we have: $m = \vec{x}(a) = ... = \vec{y}(g) = 1$). For example, to let

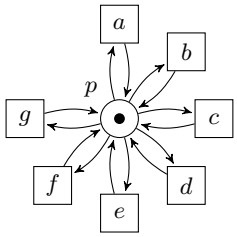



Fig. 4: Place $p$ showing flower behavior, used to encode $\overline{L}$.

[5] https://svn.win.tue.nl/repos/prom/Packages/CausalActivityMatrixMiner/Trunk/

Table 1: Basic constraints $c_1, c_2, ..., c_{11}$ of the ILP-based process discovery algorithm based on example event log $L$.

| | $m$ | $\vec{x}(a)$ | $\vec{x}(b)$ | $\vec{x}(c)$ | $\vec{x}(d)$ | $\vec{x}(e)$ | $\vec{x}(f)$ | $\vec{x}(g)$ | $\vec{y}(a)$ | $\vec{y}(b)$ | $\vec{y}(c)$ | $\vec{y}(d)$ | $\vec{y}(e)$ | $\vec{y}(f)$ | $\vec{y}(g)$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $c_1$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | −1 | 0 | 0 | 0 | 0 | 0 | 0 | $\geq 0$ | $\langle a \rangle$ |
| $c_2$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | −1 | −1 | 0 | 0 | 0 | 0 | 0 | $\geq 0$ | $\langle a, b \rangle$ |
| $c_3$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | −1 | 0 | −1 | 0 | 0 | 0 | 0 | $\geq 0$ | $\langle a, c \rangle$ |
| $c_4$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | −1 | −1 | −1 | 0 | 0 | 0 | 0 | $\geq 0$ | $\langle a, b, c \rangle$ |
| $c_5$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | −1 | −1 | −1 | 0 | 0 | 0 | 0 | $\geq 0$ | $\langle a, c, b \rangle$ |
| $c_6$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | −1 | −1 | −1 | −1 | 0 | 0 | 0 | $\geq 0$ | $\langle a, b, c, d \rangle, \langle a, c, b, d \rangle$ |
| $c_7$ | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | −1 | −1 | −1 | −1 | −1 | 0 | 0 | $\geq 0$ | $\langle a, b, c, d, e \rangle, \langle a, c, b, d, e \rangle$ |
| $c_8$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | −1 | −1 | −1 | −1 | −1 | −1 | 0 | $\geq 0$ | $\langle a, b, c, d, e, f \rangle, \langle a, c, b, d, e, f \rangle$ |
| $c_9$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | −1 | −1 | −1 | −1 | −1 | 0 | −1 | $\geq 0$ | $\langle a, b, c, d, e, g \rangle, \langle a, c, b, d, e, g \rangle$ |
| $c_{10}$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | −1 | −1 | −1 | −1 | −2 | −1 | 0 | $\geq 0$ | $\langle a, b, c, d, e, f, e \rangle, \langle a, c, b, d, e, f, e \rangle$ |
| $c_{11}$ | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 0 | −1 | −1 | −1 | −1 | −2 | −1 | −1 | $\geq 0$ | $\langle a, b, c, d, e, f, e, g \rangle, \langle a, c, b, d, e, f, e, g \rangle$ |

the Petri net in Figure 4 reproduce the empty sequence $\epsilon$, no transition needs to be fired and hence no variable is needed. For sequence $\langle a \rangle$, a token (represented by decision variable $m$) is needed which is consequently consumed by transition $a$ (represented by decision variable $\vec{y}(a)$). Thus, sequence $\langle a \rangle$ maps to the linear combination $m - \vec{y}(a)$. For $\langle a, b \rangle$, after a token is consumed by transition $a$, the token needs to explicitly be produced by transition $a$ (represented by $\vec{x}(a)$) in order for transition $b$ to be able to consume it again (represented by $\vec{y}(b)$), leading to $m - \vec{y}(a) + \vec{x}(a) - \vec{y}(b)$. Repeating this rationale for each sequence present in $\overline{L}$ yields the constraint body as depicted in Table 1.

Each linear inequality has a right hand side greater or equal to zero, specifying the requirement that the sequence should be reproducible by the place we would like to add. Again consider place $p_5$ of the Petri net shown in Figure 3 for which we have $\vec{x}(c) = 1$ and $\vec{y}(d) = 1$. Note that for each inequality presented in Table 1, $\vec{x}(c) + \vec{y}(d) \geq 0$ and thus we may accept $p_5$. Some of the sequences in $\overline{L}$ map to the same linear inequality, i.e. constraints $c_6, c_7, ..., c_{11}$ in Table 1 all have two corresponding sequences in $\overline{L}$. Thus, the linear inequalities abstract from the notion of order within the sequence, except for the last activity, i.e. constraints $c_2$ and $c_3$ differ only based on the last activity in the sequence.

The addition of trace $\langle a, b, d, e, f, e, g \rangle$ to $L$, resulting in $L'$ leads to the fact that place $p_5$ in Figure 3 is no longer accepted as a place. Due to the presence of $\langle a, b, d, e, f, e, g \rangle$ in $L'$ all its prefixes are present in $\overline{L'}$. To encode $\overline{L}$ we extend the set of linear inequalities as depicted in Table 1 with the new inequalities corresponding to the prefixes of $\langle a, b, d, e, f, e, g \rangle$ as depicted in Table 2. Note that $\langle a \rangle$ and $\langle a, b \rangle$ are already part of $\overline{L}$ and therefore do not generate new inequalities. Also note that for each inequality in Table 2 place $p_5$ evaluates to $-1$, i.e. place $p_5$ is encoded as $\vec{x}(c) + \vec{y}(d)$ and as we can see in Table 2 $\vec{x}(c) + \vec{y}(d) = -1$ for constraints $c_{12}$, $c_{13}$, ..., $c_{16}$. Thus place $p_5$ does not adhere to the newly added inequalities.

It is possible to require each place to be empty after trace completion [4, Section 4.5.2]. Enforcing emptiness after trace completion guarantees that the empty-marking is a reachable final marking of the Petri net constructed. We enforce emptiness after trace completion by specifying for each trace in $L$ ($L'$ respectively) that the corre-

Table 2: Constraints $c_{12}$, $c_{13}$,..., $c_{16}$ based on the prefixes of $\langle a, b, d, e, f, e, g\rangle$ which together with the constraints depicted in Table 1 form the constraint body for $L'$.

| | $m$ | $\vec{x}(a)$ | $\vec{x}(b)$ | $\vec{x}(c)$ | $\vec{x}(d)$ | $\vec{x}(e)$ | $\vec{x}(f)$ | $\vec{x}(g)$ | $\vec{y}(a)$ | $\vec{y}(b)$ | $\vec{y}(c)$ | $\vec{y}(d)$ | $\vec{y}(e)$ | $\vec{y}(f)$ | $\vec{y}(g)$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $c_{12}$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | −1 | −1 | 0 | −1 | 0 | 0 | 0 | $\geq 0$ | $\langle a, b, d\rangle$ |
| $c_{13}$ | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | −1 | −1 | 0 | −1 | −1 | 0 | 0 | $\geq 0$ | $\langle a, b, d, e\rangle$ |
| $c_{14}$ | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | −1 | −1 | 0 | −1 | −1 | −1 | 0 | $\geq 0$ | $\langle a, b, d, e, f\rangle$ |
| $c_{15}$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | −1 | −1 | 0 | −1 | −2 | −1 | 0 | $\geq 0$ | $\langle a, b, d, e, f, e\rangle$ |
| $c_{16}$ | 1 | 1 | 1 | 0 | 1 | 2 | 1 | 0 | −1 | −1 | 0 | −1 | −2 | −1 | −1 | $\geq 0$ | $\langle a, b, d, e, f, e, g\rangle$ |

sponding linear inequality should equal 0. For event log $L$ this would lead to adding the following constraints to the ILP's constraint body:

| $m$ | $\vec{x}(a)$ | $\vec{x}(b)$ | $\vec{x}(c)$ | $\vec{x}(d)$ | $\vec{x}(e)$ | $\vec{x}(f)$ | $\vec{x}(g)$ | $\vec{y}(a)$ | $\vec{y}(b)$ | $\vec{y}(c)$ | $\vec{y}(d)$ | $\vec{y}(e)$ | $\vec{y}(f)$ | $\vec{y}(g)$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | −1 | −1 | −1 | −1 | −1 | 0 | −1 | $= 0$ | $\langle a, b, c, d, e, g\rangle, \langle a, c, b, d, e, g\rangle$ |
| 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | −1 | −1 | −1 | −1 | −2 | −1 | −1 | $= 0$ | $\langle a, b, c, d, e, f, e, g\rangle, \langle a, c, b, d, e, f, e, g\rangle$ |

Constraints $c_{12}$, $c_{13}$, ..., $c_{16}$ depicted in Table 2 lead to rejection of place $p_5$. Thus in order to make place $p_5$ an acceptable solution we need means to filter out those constraints. We present two techniques, slack variable filtering and sequence encoding filtering. Slack variable filtering introduces a new set of variables that allow the ILP solver to ignore some constraints within the constraint body. Sequence encoding filtering leaves out constraints that refer to low-frequent exceptional behavior before actually solving the ILP problems.

### 4.1 Slack Variable Filtering

The main rationale of slack variable[6] filtering is to overcome the influence of exceptional behavior by adding the ability to ignore unwanted constraints. This is achieved by adding a slack variable for each constraint that is based on a sequence in $\overline{L'}$. Each slack variable is allowed to either have value 0 or value 1. If a slack variable is assigned a value 1, the constraint corresponding to the slack variable is ignored. To prevent the ILP from using every slack variable and effectively ignore all constraints, an additional constraint is added specifying an upper-bound w.r.t. the total number of slack variables the ILP is allowed to assigned a value of 1. The threshold value that specifies how many slack variables are allowed to use is a parameter of the approach.

In our example based on $L'$ the basic set of linear inequalities is the set $c_1, c_2, ...,$ $c_{16}$ depicted in Tables 1 and 2. Hence in slack variable based filtering we add 16 new variables $s_1$, $s_2$, ..., $s_{16} \in \{0, 1\}$, where $s_1$ corresponds to $c_1$, $s_2$ corresponds to $c_2$, ... and $s_{16}$ corresponds to $c_{16}$. In order to make $p_5$ an acceptable solution again constraints $c_{12}$, $c_{13}$, ..., $c_{16}$ should be ignored, i.e. $s_{12}$, $s_{13}$, ..., $s_{16}$ should be set to 1. Thus a suitable upper-bound for the number of slack variables that we accept to use in this case is five.

[6]Note that the term slack variable within this context refers to the slack introduced within the ILP w.r.t. filtering. In terms of terminology the term slack-variable therefore differs from the notion of conventional slack variables within the field of optimization.

Table 3: Core constraint body of the slack variable filtering ILP-based process discovery formulation applied on the running example.

| | $m$ | $\vec{x}(a)$ | $\vec{x}(b)$ | $\vec{x}(c)$ | $\vec{x}(d)$ | $\vec{y}(a)$ | $\vec{y}(b)$ | $\vec{y}(c)$ | $\vec{y}(d)$ | $s_{1_1}$ | $s_{1_2}$ | $s_{1_3}$ | $s_{1_4}$ | $s_{1_5}$ | $s_{1_6}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $c_{1_1}$ | 1 | 0 | 0 | 0 | 0 | $-1$ | 0 | 0 | 0 | $N_{1_1}$ | 0 | 0 | 0 | 0 | 0 | $\geq$ | 0 | $\langle a \rangle$ |
| $c_{1_2}$ | 1 | 1 | 0 | 0 | 0 | $-1$ | $-1$ | 0 | 0 | 0 | $N_{1_2}$ | 0 | 0 | 0 | 0 | $\geq$ | 0 | $\langle a, b \rangle$ |
| $c_{1_3}$ | 1 | 1 | 0 | 0 | 0 | $-1$ | 0 | $-1$ | 0 | 0 | 0 | $N_{1_4}$ | 0 | 0 | 0 | $\geq$ | 0 | $\langle a, c \rangle$ |
| $c_{1_4}$ | 1 | 1 | 1 | 0 | 0 | $-1$ | $-1$ | 0 | $-1$ | 0 | 0 | 0 | $N_{1_4}$ | 0 | 0 | $\geq$ | 0 | $\langle a, b, d \rangle$ |
| $c_{1_5}$ | 1 | 1 | 0 | 1 | 0 | $-1$ | 0 | $-1$ | $-1$ | 0 | 0 | 0 | 0 | $N_{1_5}$ | 0 | $\geq$ | 0 | $\langle a, c, d \rangle$ |
| $c_{1_6}$ | 1 | 1 | 1 | 0 | 0 | $-1$ | $-1$ | $-1$ | 0 | 0 | 0 | 0 | 0 | 0 | $N_{1_6}$ | $\geq$ | 0 | $\langle a, b, c \rangle$ |
| $c_{1_7}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | $\leq$ | $t_s \times \lvert\{c_1, c_2, ..., c_6\}\rvert$ | |

As an example constraint body based on slack variable filtering, using $L_1 = [\langle a, b, d \rangle, \langle a, c, d \rangle, \langle a, b, c \rangle]$ as a basis[7], consider Table 3. It comprises of a diagonal matrix ranging over the slack variables $s_{1_1}$, $s_{1_2}$, ...,$s_{1_6}$. It is important that the coefficients of the diagonal matrix, i.e., $N_{1_1}$, $N_{1_2}$, ..., $N_{1_6}$, have a sufficiently large value assigned such that the corresponding constraint is actually ignored. The additional constraint $c_{1_7}$ enforces some upper-bound to be respected w.r.t. the total number of ignored constraints. The slack variable threshold $t_s$ specifies what portion of the slack variables may be used. In the example if $\frac{1}{6} \leq t_s < \frac{1}{3}$ is used, one constraint may be dropped out of the constraint body.

To add constraints for emptiness after completion, it is important that we enforce a *coupling* to constraints representing prefixes. We should use a negative counter weight for every prefix and change the r.h.s. of the equations from *equal to zero* to *less than or equal to zero*. For example, adding emptiness after completion to the example based on $L_1$ yields the three additional constraints:

| | $m$ | $\vec{x}(a)$ | $\vec{x}(b)$ | $\vec{x}(c)$ | $\vec{x}(d)$ | $\vec{y}(a)$ | $\vec{y}(b)$ | $\vec{y}(c)$ | $\vec{y}(d)$ | $s_{1_1}$ | $s_{1_2}$ | $s_{1_3}$ | $s_{1_4}$ | $s_{1_5}$ | $s_{1_6}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $c_{1_8}$ | 1 | 1 | 1 | 0 | 1 | $-1$ | $-1$ | 0 | $-1$ | $-N_{1_1}$ | $-N_{1_2}$ | 0 | $-N_{1_4}$ | 0 | 0 | $\leq$ | 0 | $\langle a, b, d \rangle$ |
| $c_{1_9}$ | 1 | 1 | 0 | 1 | 1 | $-1$ | 0 | $-1$ | $-1$ | $-N_{1_1}$ | 0 | $-N_{1_3}$ | 0 | $-N_{1_5}$ | 0 | $\leq$ | 0 | $\langle a, c, d \rangle$ |
| $c_{1_{10}}$ | 1 | 1 | 1 | 1 | 0 | $-1$ | $-1$ | $-1$ | 0 | $-N_{1_1}$ | $-N_{1_2}$ | 0 | 0 | 0 | $-N_{1_6}$ | $\leq$ | 0 | $\langle a, b, c \rangle$ |

In the previous example case and in the case of event log $L'$ we are able to explicitly indicate an upper-bound for the number of slack variables to be used and we know what slack variables to use in order to get the result we are looking for. In practice this is not the case as the question remains what slack variables the ILP-solver will set to 1. As is, the solver can ignore constraints at random when solving a slack variable filtering-based ILP. The solver can however be steered to ignore certain constraints by manipulation of the ILP's objective function. The exact values for the object function's coefficients of the slack variables is therefore a second parameter of the approach. The effect of these coefficients is inseparable from the original objective function used as it influences a potential place's objective value.

Although slack variable filtering is a usable technique from a theoretical perspective, it has some downsides from a practical perspective. Even if we use a suitable ILP-objective function that steers the filtering to some extent, in general the ILP-solver

[7]The use of $L_1$ is mainly motivated by space constraints, i.e. the constraint body tends to be large on larger examples.

is still free in choosing what slack variables to use. In practice, when applying discovery using causal relations, the actual slack variables used may differ per place found. Hence, some places will be based on a different body of linear inequalities than others. Secondly, estimating a suitable objective function is a cumbersome task as it is partly coupled to the original objective function used. Thirdly, the computation time needed to solve an ILP is theoretically exponential in the number of variables used. Adding slack variables based on the encoding of $\overline{L}$ quickly generates ILP's with a very large amount of additional variables (worst case: $|\overline{L}|$ variables) and thus an infeasible computation time. During experimentation the slack variable filtering technique already resulted in infeasible computation time on a log only consisting of 12 event classes and a total of 1000 traces ($\pm$ 4 hours to solve all ILPs constructed). Thus, from a practical perspective the technique is of little use and, hence, we look for alternative filtering approaches.

### 4.2 Sequence Encoding Filtering

In sequence encoding filtering, like in slack variable filtering, we try to filter by ignoring constraints. Filtering within this technique is based on the encoded representation of sequences in $\overline{L}$, i.e. the linear inequalities. The fundamental difference w.r.t. slack variable filtering is the fact that in sequence encoding filtering we leave the constraints out of the constraint body. Additionally the filtering is executed one time and hence every ILP that is eventually solved uses the same set of linear constraints within its constraint body.

Let us reconsider $L' = [\langle a,b,c,d,e,g\rangle^{10}, \langle a,c,b,d,e,g\rangle^{10}, \langle a,b,c,d,e,f,e,g\rangle^{10}, \langle a,c,b,d,e,f,e,g\rangle^{10}, \langle a,b,d,e,f,e,g\rangle]$ and correspondingly the multi-set representation of it's prefix-closure i.e. $\overline{L'} = [\epsilon^{41}, \langle a\rangle^{41}, \langle a,b\rangle^{21}, \langle a,c\rangle^{20}, \langle a,b,c\rangle^{20}, \langle a,c,b\rangle^{20}\langle a, b,d\rangle, ..., \langle a,b,c,d,e,f,e,g\rangle^{10}, \langle a,c,b,d,e,f,e,g\rangle^{10}, \langle a,b,d,e,f,e,g\rangle]$. The core of sequence encoding filtering is a directed acyclic graph where each linear inequality based on a sequence in $\overline{L'}$ acts as a vertex. An example of such graph, based on $L'$ and $\overline{L'}$ is depicted in Figure 5.

The empty sequence $\epsilon$ always acts as a root vertex. A vertex representing some linear inequality $c_i$ has outgoing arcs to those vertices that represent a linear inequality of which the sequence corresponding to $c_i$ *could* act as a prefix. The arcs are labeled

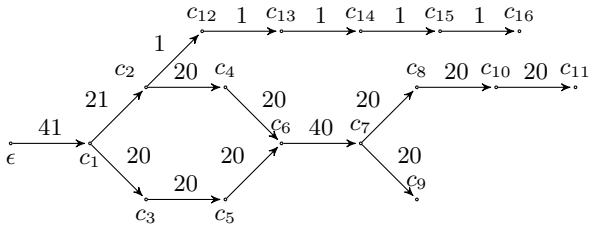



Fig. 5: Sequence encoding filtering graph based on example log $L'$ and $\overline{L'}$, where each $c_i \in \{c_1, c_2, ..., c_{16}\}$ corresponds to the linear inequalities presented in Tables 1 and 2.

using actual sequence frequencies present in the multi-set abstraction of the prefix-closure of the log. In the example depicted in Figure 5, $\epsilon$ is always followed by $c_1$, i.e. $\langle a \rangle$ is the only 1-sized sequence in $\overline{L}$. $c_1$ has two connected vertices, being $c_2$, representing $\langle a, b \rangle$ and $c_3$ representing $\langle a, c \rangle$. The cardinality of $\langle a, b \rangle$ is 21 in $\overline{L'}$ and hence the arc from $c_1$ to $c_2$ has value 21.

After constructing the filtering graph, we traverse the graph in a breadth first manner and cut off branches that represent exceptional behavior. We start at its root and assess what outgoing arcs from the root have a too low arc weight given some decision function. Once we have decided what outgoing arcs should remain we traverse each of these arcs. From the end-point of such arc, i.e. a vertex representing a constraint, we again evaluate all the outgoing arcs (if any). Only those constraints that correspond to a vertex in the filtered sequence encoding graph will be added to the ILP constraint body. The decision function that helps us in deciding whether we cut off a certain branch in the graph is a parameter of the approach.

For the implementation of the algorithm in the process mining framework ProM we have adopted the following approach. For each vertex that is the end-point of an edge that we keep we always include the outgoing edge with the maximum edge label value. Additionally we include all other edges $e$ that have a lower (or equal) value than (to) the maximum value, as long as the difference of $e$'s value w.r.t. the maximum is within some bounded range. The bounded range is typically some fraction of the maximum, this fraction is deemed the cut-off coefficient $c_c$.

As an example we apply this technique on the graph depicted in Figure 5 with $c_c = \frac{1}{10}$ of which the result is depicted in Figure 6. The root has one arc and thus we keep this arc. Traversing the arc leads us to vertex $c_1$ which has two outgoing arcs. The outgoing arc from $c_1$ with the maximum label is the arc to $c_2$ and is labeled 21. This arc will be kept in the graph. The bounded range for any other arc starting from vertex $c_1$ is now computed by multiplying the cut-off coefficient with the maximum value for this node, i.e. the bounded range is $\frac{1}{10} \times 21 = 2.1$. Any edge going out of $c_1$ that has a value greater than or equal to $21 - 2.1 = 18.9$ is kept in the graph. In this case the arc from $c_1$ to $c_3$ will also remain as it has a value of 20, which is greater than 18.9. In vertex $c_2$ we identify that we keep the edge to $c_4$, which has the maximum label. We only keep

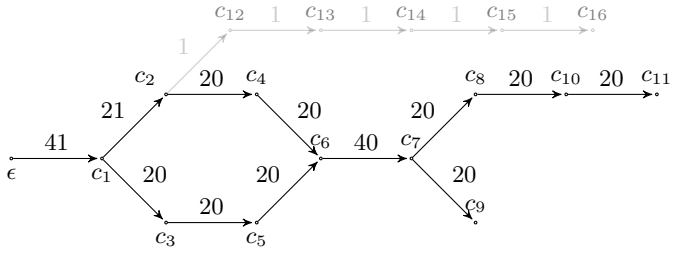



Fig. 6: Sequence encoding filtering graph based on example log $L'$ and $\overline{L'}$, where each $c_i \in \{c_1, c_2, ..., c_{16}\}$ corresponds to the constraints presented in Tables 1 and 2. Filtering affected the branch starting at $c_2$ and ending in $c_{16}$.

outgoing arcs from $c_2$ with a label value greater than or equal to $20 - \frac{1}{10} \times 20 = 18$. As a result we will drop the edge to $c_{12}$ as it only has a label value of 1. For node $c_3$ we identify only one arc leading to $c_5$ which we decide to keep. The next vertices to consider for evaluation are $c_4$ and $c_5$. Note that $c_{12}$ will never be evaluated and as a consequence neither will $c_{13}$, $c_{14}$, $c_{15}$ and $c_{16}$. Using the aforementioned approach, only constraints $c_1$, $c_2$, ..., $c_{11}$ remain part of the graph. Therefore the constraint body as depicted in Table 1 will be the result of applying sequence encoding filtering on $L'$. As a consequence, place $p_5$ in Figure 3 becomes a feasible place again.

The result of applying sequence encoding filtering on the event log used for discovery of the models depicted in Figure 2, using the aforementioned cut-off strategy with a $c_c$ value of $\frac{3}{4}$ is depicted in Figure 7. The Petri net depicted in Figure 7 shows that using sequence encoding filtering only those linear inequalitys remain in the constraint body that are related to frequent behavior whereas a great part of the exceptional behavior is successfully filtered-out.

## 5 Evaluation

Both filters have been implemented in the *HybridILPMiner*[8] package within the *ProM framework* (`http://www.promtools.org`). Using the implementation we have validated the sequence encoding filter approach against the conventional ILP-based process discovery algorithm. We evaluated model quality in terms of fitness and precision as well as the efficiency of applying sequence encoding filtering.

All experiments have been conducted using a *Dell Latitude E55400, Intel(R) Core (TM) i7-4600U CPU @ 2.10 Ghz – 2.70 Ghz (x-64), 8.00 GB RAM* laptop. As indicated in Section 4.1, from a practical perspective the slack variable filtering technique is of little use and is therefore left out of empirical evaluation. All data artifacts resulting from

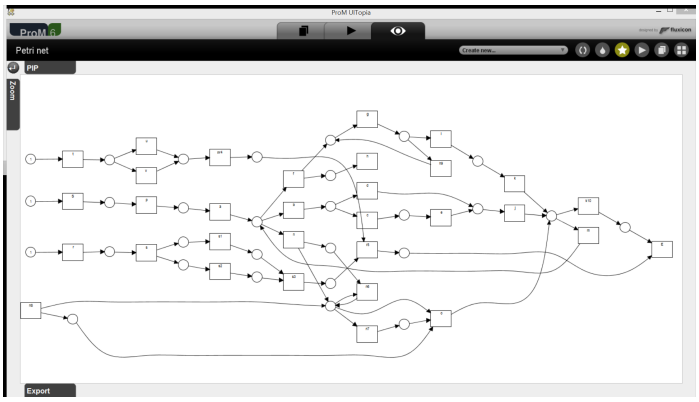



Fig. 7: Process discovery result of the sequence encoding filtering ILP-based algorithm, with $c_c = \frac{7}{10}$, using an event log that contains a minor fraction of exceptional behavior.

[8] `https://svn.win.tue.nl/repos/prom/Packages/HybridILPMiner/Tags/2015_bpm_ilp_filtering_version-0.2.1/`

the experiments described in the upcoming sections are available at `https://svn.win.tue.nl/repos/prom/Packages/HybridILPMiner/Tags/results/results_0.2.1.tar.gz`.

### 5.1 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 behavior to rule-based approaches in process discovery [12]. The event logs contain different percentage levels of exceptional behavior and are based on three ground truth event logs without exceptional behavior. These ground truth event logs are entitled *a12f0n00*, *a22f0n00* and *a32f0n00*. The two digits behind the *a* character indicate the number of event classes present within the event log, i.e. event log a12f0n00 contains 12 different event classes. For each log a total of four new logs is generated, differing in the number of traces that have been manipulated. The percentages used for 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*[9]. Exceptional behavior within the event logs is generated by either tail/head of sequence removal, random part of sequence body removal or interchange of two randomly chosen events [12].

From an evaluation point of view, the existence of ground truth event logs which do not contain exceptional behavior is of utmost importance. 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 behavior. Precision is defined as the amount of behavior allowed by the model that is also present in the event log. Thus if all behavior allowed by the model is present in the event log, precision is maximal, i.e. the precision value is 1. If the model allows for behavior that is not present in the event log, precision will be lower than 1. The more behavior is allowed by the model that is not present in the event log, the lower the precision value will be. If exceptional behavior is present in an event log, the conventional ILP-based process discovery algorithm is unable to find any meaningful patterns within the event log. The derived model will, by definition, allow for all exceptional behavior present in the event log. Hence, we expect such models to have low precision when using the ground truth event 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 behavior, we expect the precision of the discovered models to be higher.

To evaluate the sequence encoding filtering approach we have applied the conventional ILP-based process discovery algorithm and three different sequence encoding filtering instantiations for each event log. We used the branch cut-off technique as described in Section 4.2 with cut-off coefficients $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$. For each event log we both learned models with and without adoption of emptiness after case completion constraints [4, Section 4.5.2]. We measured the replay-fitness and precision based on the

[9] For discovery of the Petri net in Figure 2a we used *a32f0n00* as an input whereas for the Petri nets in Figures 2b and 7 we used *a32f0n05*.

ground truth event logs. The results of the experiments are presented in Figure 11 on Page 19..

Replay-fitness of the discovered models w.r.t. the ground truth event logs using all four approaches remains 1 in all cases[10]. Due to the incapability of handling exceptional behavior of the conventional algorithm, as expected, precision drops rapidly. For the sequence encoding filtering we identify the $\frac{1}{4}$ variant to outperform the other two. This is explained by the fact that the $c_c = \frac{1}{4}$ variant is the most rigorous filter and hence removes the most constraints. It is clear that the decrease of precision for the sequence encoding based approaches is less severe compared to the conventional approach. This is in line with the rationale as presented before as we expect the filtering based approaches to be more able in handling exceptional behavior. Therefore, we conclude that the filtering based models discover Petri net patterns that more accurately represent the dominant behavior in the input event log. Thus, the newly presented techniques allow us to successfully apply filtering whilst using ILP-based process discovery as a basis.

### 5.2 Computation time

The core of sequence encoding filtering is leaving out constraints that are likely to refer to exceptional behavior. Thus we reduce the size of the core ILP constraint body. Apart from the results w.r.t. replay-fitness and precision, the sequence encoding filtering approach also allows us to discover process models faster. In Figure 8 we have depicted the average ILP solve time per event log used in the model quality analysis. For each event log a set of causal relations was calculated each generating an ILP. We measured the solve time of each generated ILP in both the non-filtered and the filtered cases. Solving each ILP, for each filter level and the non-filtered version, was performed 50 times.

The results in Figure 8 show that cutting out some of the constraints related to exceptional behavior often leads to remarkably faster solve times. The results also show that in most cases cutting away more branches, i.e. using a lower cut-off coefficient, leads to lower computation times. However, when the amount of exceptional behavior increases, the average solve times of the filter based instantiations can exceed the average solve time of the conventional approach, i.e., a22f0nXX with 50% and af32f0nXX with 20% and 50%. Manual inspection of the models associated with the results showed that the filtered instantiations yielded models comprising of much more Petri net places than the models returned by the conventional algorithm. Thus, the filtered instances lead to more feasible ILP solutions. To quantify this relationship we have depicted the number of places found per event log in Figure 9.

For the 0% logs, all versions find the same number of places. For the other levels of exceptional behavior, surprisingly, all filter approaches find the exact same number of places whereas the conventional approach finds less. Thus, the $\frac{3}{4}$ variant already seems to filter out all constraints that cause the conventional approach to be unable to find places. Figure 10 depicts the solve times relative to the number of places found.

The results show that the sequence encoding filtering instantiations have lower solve times relative to the number of places found. In case of a12f0nXX with noise level 50%

[10] One exception for SEF with $c_c = \frac{1}{2}$, where replay fitness equals $0.99515$.

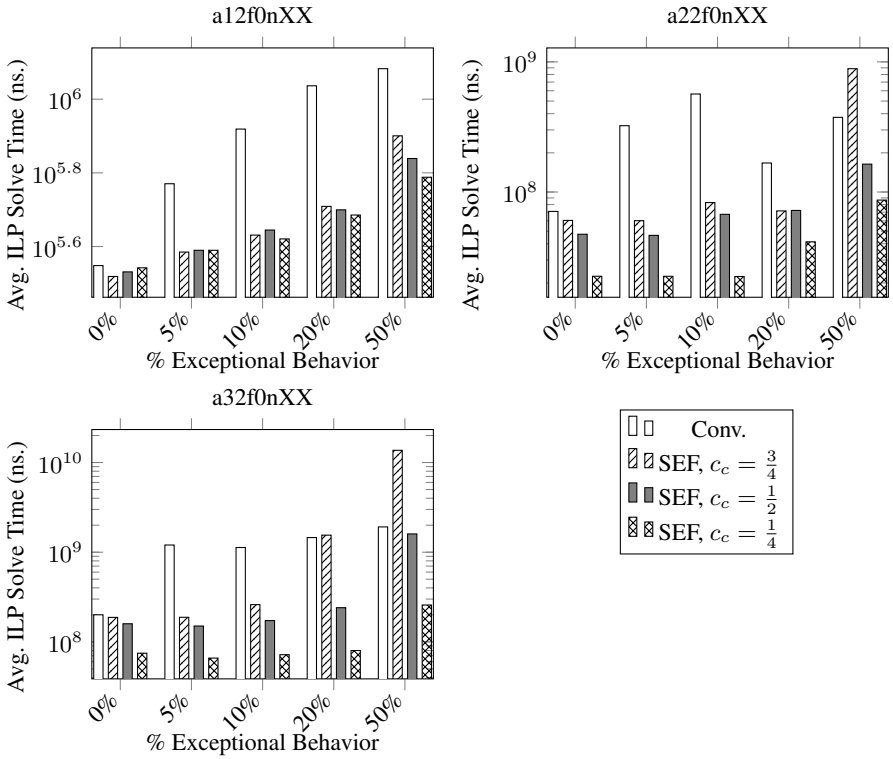



Fig. 8: Average solve time (ns.) (avg. of 50 runs) for solving an ILP based on the conventional ILP-based process discovery algorithm and the sequence encoding filtering approach with $c_c = \frac{1}{4}$, $c_c = \frac{1}{2}$ and $c_c = \frac{3}{4}$.

the conventional algorithm did not find any place and hence the data point is left out of the chart. In general it seems that due to the high level of exceptional behavior, the conventional algorithm ends up trying to solve a large number of ILPs that do not have any feasible solution. The filtered variants seem to actually do find solutions for the corresponding filtered ILPs. In some of these cases, the underlying solver seems faster in deciding that there is no feasible solution compared to actually finding a feasible solution given a slightly smaller constraint body, i.e. using a filtered constraint body.

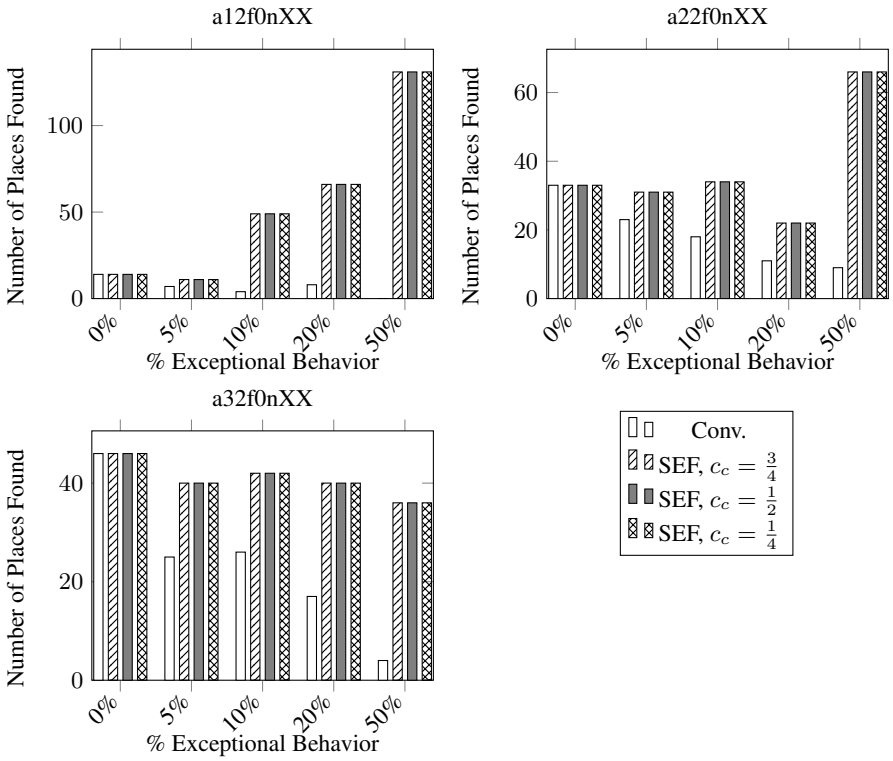



Fig. 9: Total number of places found, based on solving ILPs based on the conventional ILP-based process discovery algorithm and the sequence encoding filtering approach with $c_c = \frac{1}{4}$, $c_c = \frac{1}{2}$ and $c_c = \frac{3}{4}$.

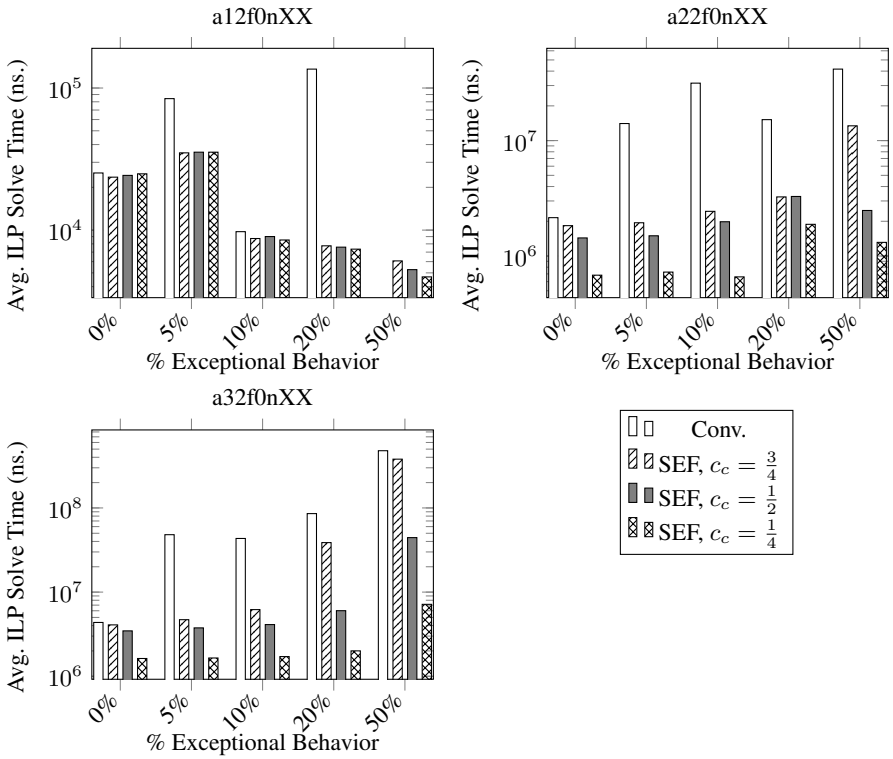



Fig. 10: Average solve time (ns.) (avg. of 50 runs) relative to the number of places, i.e. feasible solutions, for solving an ILP based on the conventional ILP-based process discovery algorithm and the sequence encoding filtering approach with $c_c = \frac{1}{4}$, $c_c = \frac{1}{2}$ and $c_c = \frac{3}{4}$.

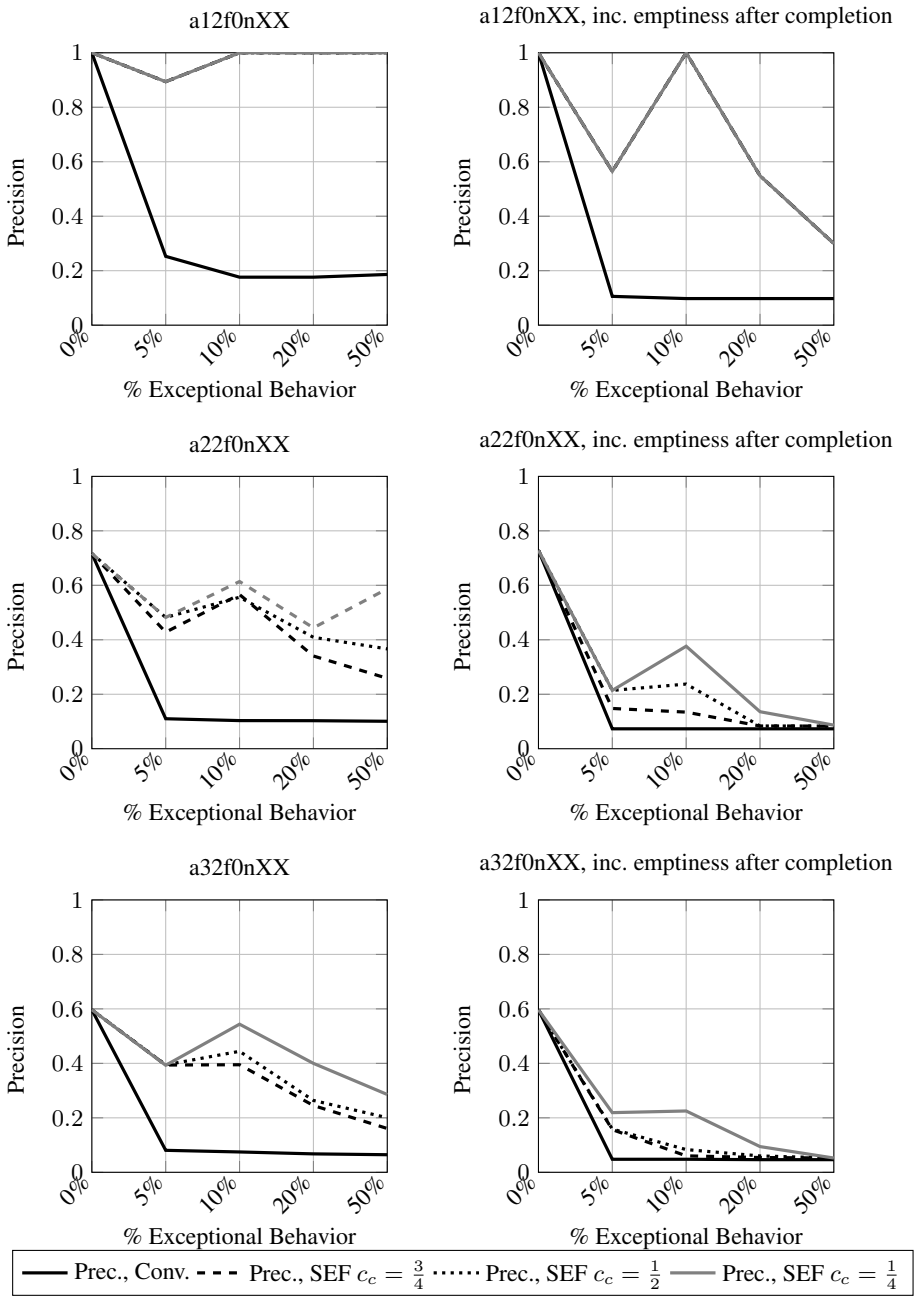



Fig. 11: Precision (Prec.) measurements based on the *a12f0nXX*, *a22f0nXX* and *a32f0nXX* logs. Process discovery based on the conventional ILP-based process discovery algorithm (Conv.) and sequence encoding filtering (SEF) with three different cut-off coefficient values $\frac{1}{4}$, $\frac{1}{2}$ and $\frac{3}{4}$.

## 6 Conclusion

The work presented in this paper is motivated by the observation that all existing region-based techniques are unable to cope with exceptional low-frequent behavior in event logs. ILP based process discovery has several advantages, but the inability to abstract from infrequent exceptional behavior makes it unusable in real-life settings. We presented two techniques that enable us to apply filtering exceptional behavior within the ILP-based process discovery algorithm. Both techniques allow us to find models with acceptable trade-offs w.r.t. replay fitness and precision. However, only sequence encoding filtering is feasible for larger event logs. We showed that sequence encoding filtering enables us to find Petri net structures in data consisting of exceptional behavior, using ILP-based process discovery as an underlying technique. An additional benefit of the approach is that it often decreases the average time needed to solve the corresponding ILP problems. However, in some cases the technique might lead the underlying solver to find more feasible ILP solutions w.r.t. the conventional approach, which might lead to higher solve times. This only seems the case if the cut-off coefficient chosen is relatively high, i.e. closer to 1.

The other technique presented, being slack-variable based filtering, is theoretically able to handle event logs consisting of exceptional behavior. From a practical perspective the techniques is not feasible as it generates ILP problems with too high computation times.

An interesting direction for future research is assessing the combination of ILP-based filtering techniques and event log decomposition [13, 14]. Specifically slack-variable based filtering might benefit from integration with decomposition as the ILP problems constructed in a decomposed setting are typically smaller than the ILP problems constructed in a traditional event log based setting.

## References


1. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F., eds.: On the Move to Meaningful Internet Systems: OTM 2012. Volume 7565 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 305–322
3. Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. Acta Inf. **33**(4) (1996) 297–315
4. Werf, J.M.E.M.v.d., Dongen, B.F.v., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundamenta Informaticae **94**(3) (2009) 387–412
5. Leemans, S.J.J., Fahland, D., Aalst, W.M.P.v.d.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings. (2013) 311–329
6. Leemans, S.J.J., Fahland, D., Aalst, W.M.P.v.d.: Process and Deviation Exploration with Inductive Visual Miner. In: Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014), Eindhoven, The Netherlands, September 10, 2014. (2014) 46

7. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
8. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France. (2011) 310–317
9. Günther, C.W., Aalst, W.M.P.v.d.: Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings. (2007) 328–343
10. Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: A Genetic Algorithm for Discovering Process Trees. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012. (2012) 1–8
11. Wiel, T.v.d.: Process mining using integer linear programming. Master's thesis, Eindhoven University of Technology (2010)
12. Maruster, L., Weijters, A.J.M.M., Aalst, W.M.P.v.d., Bosch, A.v.d.: A Rule-Based Approach for Process Discovery: Dealing with Noise and Imbalance in Process Logs. Data Min. Knowl. Discov. **13**(1) (2006) 67–87
13. Aalst, W.M.P.v.d.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases **31**(4) (2013) 471–507
14. Verbeek, H.M.W., Aalst, W.M.P.v.d.: Decomposed process mining: The ilp case. In: BPM 2014 Workshops. Volume 202. Springer (2015) 264–276