Discovering Block-Structured Process Models From Event Logs Containing Infrequent Behaviour

Sander J.J. Leemans, Dirk Fahland, and Wil M.P. van der Aalst

Eindhoven University of Technology, the Netherlands
{s.j.j.leemans, d.fahland, w.m.p.v.d.aalst}@tue.nl

Abstract Given an event log describing observed behaviour, process discovery aims to find a process model that 'best' describes this behaviour. A large variety of process discovery algorithms has been proposed. However, no existing algorithm returns a sound model in all cases (free of deadlocks and other anomalies), handles infrequent behaviour well and finishes quickly. We present a technique able to cope with infrequent behaviour and large event logs, while ensuring soundness. The technique has been implemented in ProM and we compare the technique with existing approaches in terms of quality and performance.

Keywords: process mining, process discovery, block-structured process models, soundness, fitness, precision, generalisation

1 Introduction

Process mining techniques aim to support organisations in improving their business processes. Event logs of historical behaviour can be used to discover process models of the real processes as present in the organisation, as opposed to manually created models that reflect wishful thinking, should-be or as-it-was-two-years-ago behaviour. Auditing of discovered models can prove compliance with organisational and governmental regulations [3], and replay of historical behaviour on the discovered model can reveal social networks and bottlenecks [17,15,4].

The challenge in process discovery is to find the 'best' process model given recorded historical behaviour. Which process model is 'best' is typically evaluated using several quality criteria. Four important quality criteria are fitness, precision, generalisation and simplicity. An un*fitting* model cannot reproduce all behaviour recorded in the log. An im*precise* model allows for too much additional behaviour that is not described in the log. A non-*general* model only describes the behaviour in the log and therefore might disallow future behaviour absent in the log. A non-*simple* model needs a lot of places, transitions and arcs to express its behaviour and might be hard to read.

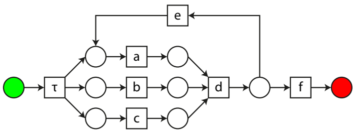


Figure 1: Unsound process model.

Another important quality criterion is *soundness*: all process steps can be executed and some satisfactory end state, the *final marking*, is always reachable. For instance, the

Petri net in Figure 1 is not sound as it contains a deadlock from which the final marking with only a single token in the final place can never be reached. An unsound process model can still be useful, but applying tasks such as evaluation, auditing, finding social networks and bottlenecks can be difficult if not impossible. Therefore, for most use cases an unsound process model can be discarded without even considering the event log it is supposed to represent.

Traces in a log might follow many different paths through the process. In most real-life event logs, some paths are taken infrequently, or traces only differ by occurrence of infrequent activities. Such logs contain *infrequent behaviour* and challenge discovery algorithms, as a process model scoring well on all quality criteria might not exist. If infrequent behaviour is included in the model, simplicity might be sacrificed, if infrequent behaviour is excluded from the model, fitness might be sacrificed. Fortunately, the Pareto principle (also known as the 8020 rule) often applies to event logs. Typically, 80% of the observed behaviour can be explained by a model that is only 20% of the model required to describe all behaviour. The *80% model* shows the "highways" in the process. Hence, it is more intuitive, but can also be used as a starting point for outlier detection. [2].

To obtain an 80% model, a classical approach is to globally filter the log before discovering a model. This has numerous disadvantages, as it is difficult to identify infrequent behaviour, and even when infrequent behaviour is filtered out, discovery algorithms (α [6], B' [16], ILP [21]) might still produce undesirable models. Other approaches were designed to ignore infrequent behaviour and can produce an 80% model but may perform less on other quality criteria: genetic approaches [5,9] have long run times and a heuristic approach [20] produces unsound models.

As of today, no technique has been proposed that *discovers a sound 80% model, does that fast and is able to filter infrequent behaviour*. Several existing approaches apply divide-and-conquer techniques [10,22,16], in which the event log is split and a model is constructed recursively. In this paper we present an extension of such an approach, IM, called *Inductive Miner - infrequent* (IMi), that aims to discover a sound 80% model fast. We introduce infrequent behaviour filters in all steps of IM, such that infrequent behaviour is filtered locally.

IMi is implemented in the InductiveMiner package of the ProM framework [14]. To evaluate IMi, we compare its performance and its discovered models to other discovery algorithms by means of the quality criteria using real-life logs.

The remainder of this paper starts with a description of logs, process trees and IM. In Section 3, IMi is introduced. In Section 4 IMi is compared to existing mining algorithms. Section 5 concludes the paper.

2 Preliminaries

Event Logs. An *event log* is a collection of traces. Each *trace* is a sequence of *events* that represent occurrences of *activities* of a process execution in the respective order. Note that a trace might appear multiple times in an event log. The trace without events is denoted with ϵ.

Process trees. The block-structured process models discovered by IM, ETM and IMi are process trees. A *process tree* is an abstract representation of a sound block-structured workflow net [6]. A tree represents a language, a leaf describes the singleton language of an activity, and a non-leaf node is an operator that describes how the languages of its children are combined. In this paper, we will consider four operators: $\times$, $\rightarrow$, $\wedge$ and $\circlearrowleft$. The $\times$ operator denotes the exclusive choice between its children, $\rightarrow$ the sequential composition and $\wedge$ the interleaved parallel composition. The $\circlearrowleft(m_1, m_2 \dots m_n)$ has two groups of children: m_1 is the loop *body* and $m_2 \dots m_n$ is the loop *redo* part. A trace in the language of $\circlearrowleft(m_1, m_2 \dots m_n)$ starts with a trace from m_1, followed by a repetition of a trace from any $m_2 \dots m_n$ and a trace from m_1 again. For instance, the language of $\circlearrowleft(a, b, c)$ is $\{\langle a\rangle, \langle a, b, a\rangle, \langle a, c, a\rangle, \langle a, b, a, c, a\rangle \dots\}$.

Another example of a process tree is $\times(\rightarrow(a, b), \wedge(c, d), \circlearrowleft(e, f))$, denoting the language $(ab)|(cd)|(dc)|(e(fe)^*)$. For a formal definition, please refer to [16].

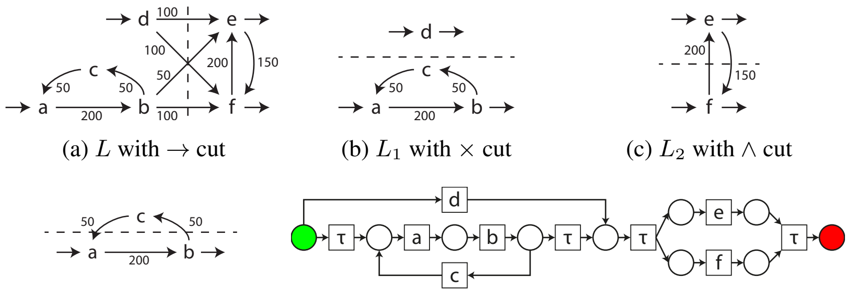
Inductive Miner In this paper, we extend an existing divide-and-conquer approach to process discovery. Divide-and-conquer has been used in process discovery before. For instance, [**?**] combines it with transition systems and regions; [22] combines it with trace alignments. In this paper we extend the Inductive Miner (IM) [16], of which we first give its basic algorithmic idea and illustrate it with a running example.

IM works by recursively a) selecting the root operator that best fits L, b) dividing the activities in log L into disjoint sets and c) splitting L using these sets into sublogs. These sublogs are then mined recursively, until a sublog contains just a single activity. We first introduce how IM selects an operator and an activity division, and illustrate it with a running example.

Consider log L: $[\langle a, b, c, a, b, e, f\rangle^{50}, \langle a, b, f, e\rangle^{100}, \langle d, e, f\rangle^{100}, \langle d, f, e\rangle^{100}]$. In a *directly-follows graph*, each node represents an activity and an edge from node a to node b is present if and only if a is directly followed by b somewhere in L. The *frequency* of edge (a, b) is how often this happens. Figure 2a shows the directly-follows graph of L. IM searches for a characteristic division of activities into disjoint sets, a *cut*, of the directly-follows graph. Each operator ($\times$, $\rightarrow$, $\wedge$ or $\circlearrowleft$) has a characteristic cut of the directly-follows graph. If such a characteristic matches, IM selects the corresponding operator. Otherwise, a *flower model*, allowing for all sequences of activities, is returned.

The dashed line in Figure 2a is a $\rightarrow$ cut: all edges crossing it go from left to right. Using the cut $\{a, b, c, d\}, \{e, f\}$, IM splits the log by splitting each trace corresponding to the cut: $L_1 = [\langle a, b, c, a, b\rangle^{50}, \langle a, b\rangle^{100}, \langle d\rangle^{200}]$ for the left branch, $L_2 = [\langle e, f\rangle^{150}, \langle f, e\rangle^{200}]$ for the right branch. Then, IM recurses. We first consider L_1. Figure 2b shows its directly-follows graph, the dashed line denotes an $\times$ cut, as no edge crosses the cut. The log L_1 is split in $L_3 = [\langle a, b, c, a, b\rangle^{50}, \langle a, b\rangle^{100}]$ and $L_4 = [\langle d\rangle^{200}]$. L_4 consists of only a single activity, so for L_4 IM discovers the leaf d. The discovered process tree up till now is $\rightarrow(\times(\dots, d), \dots)$.

IM recurses further. Figure 2c shows the directly-follows graph of L_2 with its $\wedge$ cut, which splits L_2 into $L_5 = [\langle e\rangle^{350}]$ and $L_6 = \langle f\rangle^{350}]$. Figure 2d shows the directly-follows graph of L_3 with its $\circlearrowleft$ cut. IM splits L_3 into $L_7 = [\langle a, b\rangle^{200}]$ and $L_8 = [\langle c\rangle^{50}]$. The complete process tree discovered by IM is $\rightarrow(\times(\circlearrowleft(\rightarrow(a, b), c), d), \wedge(e, f))$. Figure 2e shows the corresponding Petri net. For more details, see [16].



(a) L with $\rightarrow$ cut (b) L_1 with $\times$ cut (c) L_2 with $\wedge$ cut

(d) L_3 with $\circlearrowleft$ cut (e) Discovered Petri net

Figure 2: Directly-follows graphs. Dashed lines denote cuts. Edges have their frequencies denoted. (e) is the mined Petri net.

3 Extending IM

In this section, we introduce *Inductive Miner - infrequent* (IMi) by adding infrequent behaviour filters to all steps of IM. For each of the operational steps of IM it is described how infrequent behaviour affects the step and how distinguishing frequent and infrequent behaviour can be used to improve discovery of the 80% model. In each recursion step, IMi first applies the steps of IM unaltered. Only if this fails and IM would return a flower model, the filters are applied.

Frequencies of traces and events are ignored by IM but are taken into account by IMi in order to distinguish frequent and infrequent behaviour. In the operator and cut selection steps, two techniques are applied: filtering the directly-follows graph for infrequent edges and using a variant of the directly-follows graph for selection of $\rightarrow$. Filters are added to base case detection to filter accumulated artifacts of filtering over recursions. In the following, k denotes a user-defined threshold value between 0 and 1 to separate frequent and infrequent behaviour. Filters on the operator and cut selection steps are described first, followed by filters on base cases, and filters on log splitting.

3.1 Filters on Operator & Cut Selection

In the operator and cut selection steps, a heuristics-style filter is applied by IMi. In case of $\rightarrow$, a variation of the directly-follows graph can be used.

Heuristics-style Filtering. Consider log L_1: $[\langle a, b, c, a, b, e, f\rangle^{50}, \langle a, b, f, e\rangle^{100}, \langle d, e, f\rangle^{100}, \langle d, f, e\rangle^{100}, \langle d, e, d, f\rangle^{1}]$, which is the log used in Section 2 extended with an infrequent trace $\langle d, e, d, f\rangle$. Figure 3a shows its directly-follows graph. Compared to Figure 2a, the infrequent trace introduces the edge (e, d), and therefore the dashed line is not a $\rightarrow$ cut.

Similar to the technique used in HM, IMi filters the directly-follows graph to only contain the most frequent edges. The edge (e, d) is relatively infrequent compared to the other outgoing edges of e. An outgoing edge of a node is too infrequent if it has a frequency of less than k times the frequency of the strongest outgoing edge of that node. All too infrequent edges are filtered out in IMi before cuts of $\times$, $\rightarrow$ and $\circlearrowleft$ are detected.

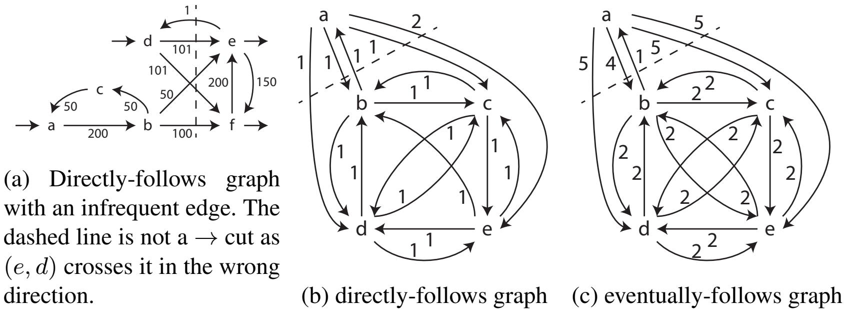
Eventually-follows Graph. Despite heuristics-style filtering, infrequent edges might remain in the directly-follows graph. Consider log $L_2 = [\langle a, c, d, e, b\rangle, \langle a, b, a, e, d, c\rangle, \langle a, e, c, b, d\rangle, \langle a, d, b, c, e\rangle]$. The second trace is the only trace containing two as: the second a is infrequent. Figure 3b shows the directly-follows graph of L_2. The dashed line in Figure 3b is not a sequence cut as edge (b, a), introduced by the infrequent a, crosses it in the wrong direction. As all outgoing edges of b have frequency 1, no value of k could filter edge (b, a).

Similar to a technique used in [19] ("weak order relation"), IMi uses the *eventually-follows graph*, which is the transitive closure of the directly-follows relation: an edge (a, b) is present if and only if a is followed by b somewhere in the log.

The eventually-follows graph of L_2 is shown in Figure 3c. In this graph, all outgoing edges of b are amplified, except the infrequent edge (b, a), which can then be filtered out.

In this example, using the eventually-follows graph allows IMi to deal with infrequent behaviour.

An infrequent occurrence of an activity still increases frequency of infrequent edges, but adds at most 1 to each of them. The eventually-follows graph amplifies all other behaviour, so using the eventually-follows graph for → cut detection increases robustness against infrequent behaviour. IMi uses a filtered eventually-follows graph to detect → cuts and if it finds one, selects → as operator.



(a) Directly-follows graph with an infrequent edge. The dashed line is not a → cut as (e, d) crosses it in the wrong direction.

(b) directly-follows graph

(c) eventually-follows graph

Figure 3: Directly and eventually follows graphs.

3.2 Filters on Base Cases

In addition to the single-activity base case in IM, as an artifact of filtering it is possible that traces without events, ϵ, remain. On both base cases filters are introduced.

Single Activities. Assume the following two logs:

$$\begin{array}{rl} L_1 = & [\epsilon^{100}, \langle a\rangle^{100}, \langle a, a\rangle^{100}, \langle a, a, a\rangle^{100}\ ] \\ L_2 = & [\epsilon^{1}\ \ \ , \langle a\rangle^{100}, \langle a, a\rangle^{1}\ \ \ , \langle a, a, a\rangle^{1}\ \ \ ] \end{array}$$

Both L_1 and L_2 consist of a single activity, cannot be split further and are base cases. Given the representational bias of IMi, for both logs either a flower model or a single activity a can be discovered. In L_1, all traces are frequent and a flower model is obviously the best choice. In L_2 however, only $\langle a \rangle$ is frequent and a best represents the frequent behaviour.

Choosing either option influences quality dimensions: discovering a for L_1 sacrifices fitness, while discovering a flower model for L_2 sacrifices precision. a is only discovered by IMi if the average number of occurrences per trace of a in the log is close enough to 1, dependent on the relative threshold k.

Empty Traces. Assume the following log: $L = [\langle a, b, d \rangle^{100}, \langle a, c, d \rangle^{100}, \langle a, d \rangle]$. In the first recursion, IMi selects the $\rightarrow$ operator and splits L into $L_1 = [\langle a \rangle^{201}]$, $L_2 = [\epsilon^1, \langle b \rangle^{100}, \langle c \rangle^{100}]$ and $L_3 = [\langle d \rangle^{201}]$.

Consider L_2. A fitting solution for the empty trace in L_2 would be to mine $\times(\tau, \ldots)$ and recurse on $L_2 \setminus \{\epsilon\}$. For L_2, ϵ is infrequent and discovering $\times(\tau, \ldots)$ would sacrifice simplicity. This is a tradeoff, but for L_2 clearly $\times(\tau, \ldots)$ is preferred. To overcome this problem, IMi only discovers $\times(\tau, \ldots)$ if ϵ is frequent enough compared to the number of traces in the log and with respect to k. If ϵ is not frequent enough, IMi filters ϵ from L_2 and recurses on $L_2 \setminus \{\epsilon\}$.

3.3 Filters on Log Splitting

Assuming the operator and cut have been selected, some infrequent behaviour in the log might not fit the chosen operator and cut. If not filtered out, this unfitting behaviour might accumulate over recursions and obscure frequent behaviour.

This section describes how infrequent behaviour can be filtered during log splitting. It is assumed that the operator and cut are correctly selected and that any behaviour that violates this selection is infrequent. For each operator, we describe the types of violations that can be detected and how they are filtered by IMi, illustrated by an example. In these examples, $\Sigma_1 = \{a\}, \Sigma_2 = \{b\}$ is the chosen cut and L_1, L_2 are the sublogs to-be-created.

- $\times$ Behaviour that violates the $\times$ operator is the presence of activities from more than one subtree in a single trace. For instance, the trace $t_1 = \langle a, a, a, a, b, a, a, a, a \rangle$ contains activities from both Σ_1 and Σ_2. Σ_1 explains the most activities, is most frequent. All activities not from Σ_1 are considered infrequent and are discarded: $\langle a, a, a, a, a, a, a, a \rangle \in L_1$.
- $\rightarrow$ Behaviour that violates the $\rightarrow$ operator is the presence of events out of order according to the subtrees. For instance, in the trace $t_2 = \langle a, a, a, a, b, b, b, b, a, b \rangle$, the last a occurs after a b, which violates the $\rightarrow$. Filtering infrequent behaviour is an optimisation problem: the trace is to be split in the least-events-removing way.
  In t_2, the split $\langle a, a, a, a \rangle \in L_1$, $\langle b, b, b, b, b \rangle \in L_2$ discards the least events.
- $\wedge$ A parallel operator allows for any sequence of behaviour of its subtrees. Therefore, no behaviour violates $\wedge$ and infrequent behaviour can be neither detected nor filtered while splitting the log.
- $\circlearrowleft$ Behaviour that violates the $\circlearrowleft$ operator is when a trace does not start or end with the loop body: For instance, $\circlearrowleft(a, b)$, is violated by all traces that do not start and end with an a. For each such invalid start or end of a trace, an empty trace is added to

L_1 to increase fitness of the resulting model. Considering the trace $t_3 = \langle b, a, b\rangle$, then $[\epsilon^2, \langle a\rangle^1] \subseteq L_1$ and $[\langle b\rangle^2] \subseteq L_2$.

In each recursion step, first the operator and cut selection steps of IM are performed by IMi. If that would result in the flower model, the procedure is applied again, with the infrequent behaviour filters in operator and cut selection, base cases and log splitting, such that in all steps of IM filters are applied by IMi. In the next section, IMi is compared to existing process discovery mining techniques.

4 Comparison to Other Discovery Algorithms

In this section, we compare IMi to existing mining algorithms on performance and quality criteria of discovered models, using ideas from [18,11]. We first describe the experimental setup and the used logs, and finish with a discussion of the results.

4.1 Experimental setup

We compare the mining algorithms IM, IMi, HM, ILP and ETM using the following quality criteria: we compare performance and measure soundness, fitness, precision, generalisation and simplicity. To provide a baseline, we include a flower model (FM), allowing for all sequences of activities. and a trace model (TM)[1]. Figure 4 gives an overview of the experimental setup.

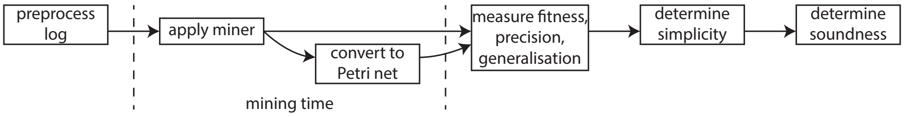


Figure 4: Experimental setup

Preprocessing. As a preprocessing step, we add artificial start and end events to the logs. Mining algorithms might require single start and end events, and these events help to determine soundness.

Mining. Secondly, the miners are applied: IM IMi, ILP, HM, ETM, FM and TM. We compare all mining algorithms using their default settings. Like in [11], parameter optimisation is outside the scope of this paper. HM and ETM do not produce a Petri net. Therefore the output of each of these miners is converted to a Petri net, measured mining time includes this conversion. We report an indicative mining time on a dual Intel Xeon E5-2630 hexacore, having 64GB of RAM, running 64-bit Windows 7. As we want to mine models fast, we set a maximum mining time of two hours. ILP is stopped abruptly after this bound, ETM is allowed to finish its round of genetic steps.

[1] A trace model allows for all traces in the event log, but no other behaviour.

Measuring. We are permissive in the soundness check: we add obvious final markings to the discovered models if the mining algorithm does not provide it, and each reasonable final marking in the discovered models is considered to be a valid final marking to be reached by the process.

To measure fitness [7], precision [8] and generalisation [1] of the mined models we use the *PNetReplayer* package in the ProM framework [14]. For these measures, first a projection of the log on the discovered model is computed, an *alignment*. The technique described in [7] provides an alignment that shows the least deviation between log and model.

For computation of the alignment, the final marking to be reached is relevant. On the models discovered by miners that do not provide a final marking, we compute the alignment assuming that every marking is a final marking. For fitness, this yields an upper bound[2]. Experiments show that the upper bound is not necessarily tight: we found differences of 0.3 in fitness between measured with and without final marking. In the results, we denote these upper bound fitness values using *italics*.

From the alignment, a graph of reached and reachable markings, and edges between them is computed. On the markings in this graph, the number of edges that is never used in the alignment is a measure for precision [8], while the frequency of the edges used in the alignment is a measure for generalisation. The values of precision and generalisation highly depend on the chosen optimal alignment. Therefore, the results with and without final marking should not be compared for precision and generalisation. Experiments show that the values are quite close: we found differences with a maximum of about 0.1 in precision when there is no final marking, and 0.005 for generalisation. We denote values obtained without final marking in *italics*.

We assess simplicity by measuring the number of arcs, places and transitions in the Petri nets.

4.2 Logs

To compare the mining algorithms, we use 12 real-life event logs. Table 1 characterises the different logs. A process from the gynaecology department of an academic hospital is logged in the BPIC'11 log [12]. The BPIC'12 log [13] originates from an application process for a personal loan or overdraft within a global financial organisation. Furthermore, we use non-public logs of a building permit approval process in five municipalities, resulting from the CoSeLog project[3]. We include these five both untouched, WABO 1 through 5, and filtered to contain only activities common to all five, WABO 1_c through 5_c.

4.3 Results

Table 1 shows the results. ✗ indicates an unsound model, ✓ a sound. A dash (-) indicates that the miner did not produce a result, an empty space indicates that measurement could not be obtained on our machine due to memory restrictions. For some experiments,

[2] We adapted the fitness computation in the PNetReplayer package to achieve this.

[3] See http://www.win.tue.nl/coselog/wiki/start.

mining took longer than two hours. This is denoted with (1). The experiments for which a final marking had to be guessed are denoted with (2).

A model with a deadlock is denoted with (10). (11) denotes that the model contains a dead transition, (12) that the model contains either an unbounded or an unreachable transition.

Table 1: Log sizes and results.

		BPIC'11	BPIC'12	WABO 1	WABO 1.c	WABO 2	WABO 2.c	WABO 3	WABO 3.c	WABO 4	WABO 4.c	WABO 5	WABO 5.c
	traces	1143	13087	434	434	286	286	481	481	324	324	432	432
	events	150291	262200	13571	9287	10439	6898	16566	11846	9743	6650	13370	8752
	activities	624	36	173	44	160	44	170	44	133	44	176	44
remarks	IM												
	IMi												
	HM	(1)	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)
	ILP	(1)		(1)				(1)	(1)	(1)			
	ETM												
	FM												
	TM	(1)	(1)										
soundness	IM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	IMi	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	HM	-	(12) ✗	(10) ✗	(11) ✗	(10) ✗	(11) ✗	(12) ✗	(12) ✗	(12) ✗	(12) ✗	(12) ✗	(12) ✗
	ILP	-	(12) ✗	-	(12) ✗		(12) ✗	-	-	-	(12) ✗		(12) ✗
	ETM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	FM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	TM	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
mining time (s)	IM	68.3	5.6	0.8	0.2	0.6	0.2	0.9	0.3	0.4	0.1	0.6	0.2
	IMi	182.3	8.1	5.1	0.4	1.7	0.6	1.8	0.5	0.8	0.6	1.3	0.6
	HM	7200.0	2519.2	1.9	0.2	2.5	2.1	1.9	0.2	1.1	0.1	2.0	2.3
	ILP	7200.0	5085.3	7200.0	319.8	1343.5	123.3	7200.0	7200.0	7200.0	1406.2	1452.4	185.1
	ETM	7220.7	51.1	7261.7	2189.1	6018.9	2539.8	5524.2	7282.5	7260.2	2998.1	4268.5	4828.4
	FM	1.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0	0.1	0.0
	TM	7200.0	7200.0	320.6	27.9	252.6	25.1	361.3	87.1	115.8	16.8	131.1	19.4
fitness	IM	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	IMi	0.698	0.931	0.756	0.780	0.977	0.888	1.000	0.874	0.993	0.757	0.990	0.833
	HM	-			*0.940*		*0.957*				*0.979*		*0.960*
	ILP	-	*1.000*	-	*1.000*		*1.000*	-	-	-	*1.000*	*1.000*	*1.000*
	ETM	0.158	0.022	0.372	0.709	0.464	0.616	0.520	0.593	0.403	0.698	0.562	0.680
	FM	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	TM	-	-		1.000		1.000		1.000		1.000		1.000
precision	IM	0.009		0.040	0.090	0.034	0.077	0.035	0.075	0.070	0.090	0.038	0.083
	IMi			0.300	0.637	0.042	0.465	0.078	0.599	0.091	0.605	0.058	0.644
	HM	-			*0.744*		*0.725*				*0.489*		*0.622*
	ILP	-	*0.306*	-	*0.537*		*0.413*	-	-	-	*0.352*	*0.324*	*0.391*
	ETM	0.927	1.000	0.937	0.913	0.973	0.994	0.920	0.952	0.961	0.890	1.000	0.895
	FM	0.002	0.051	0.011	0.043	0.009	0.037	0.010	0.040	0.012	0.040	0.009	0.039
	TM	-	-		1.000		1.000		1.000		1.000		1.000
generalisation	IM	1.000		0.999	1.000	1.000	0.999	1.000	1.000	0.999	0.999	0.999	1.000
	IMi			0.999	1.000	0.999	1.000	0.999	1.000	0.999	0.999	0.998	1.000
	HM	-			*0.998*		*0.998*				*0.923*		*0.895*
	ILP	-	*0.997*	-	*0.803*		*0.916*	-	-	-	*0.824*	*0.690*	*0.784*
	ETM	1.000	1.000	1.000	0.999	1.000	1.000	1.000	0.998	1.000	0.992	1.000	0.992
	FM	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999	1.000
	TM	-	-		0.046		0.039		0.035		0.050		0.044

4.4 Discussion

First observation is that for all logs a model was discovered within two hours by IM, IMi, FM and ETM. IMi was for all logs a bit slower than IM, while taking a lot less time

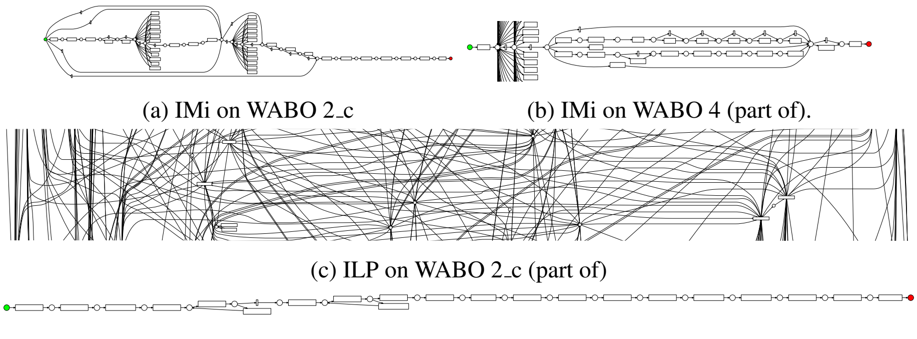
Table 2: Simplicity (#arcs,#places,#transitions).

	IM	IMi	HM	ILP	ETM	FM	TM
BPIC'11	1256,5,628	1290,27,645	-	-	16,7,8	1256,3,628	-
BPIC'12	80,7,40	166,41,81	90375,76,16737	919,88,38	2,2,1	80,3,40	-
WABO 1	368,12,184	474,97,237	1071,350,496	-	32,15,16	354,3,177	19582,9513,9791
WABO 1_c	96,5,48	122,37,61	249,92,121	802,73,46	72,32,35	96,3,48	6056,2890,3028
WABO 2	336,8,168	406,43,202	870,324,419	4560,215,162	56,28,26	328,3,164	17340,8454,8670
WABO 2_c	120,16,60	112,25,56	235,92,116	770,62,46	44,21,22	96,3,48	5754,2766,2877
WABO 3	378,18,189	358,14,179	946,344,459	-	50,24,23	348,3,174	20128,9813,10064
WABO 3_c	122,15,61	116,32,58	279,92,135	-	42,21,18	96,3,48	10604,5116,5302
WABO 4	304,23,152	290,21,145	764,270,362	-	34,16,17	274,3,137	12186,5907,6093
WABO 4_c	100,7,50	108,33,54	230,92,114	1239,80,46	78,35,31	96,3,48	4742,2255,2371
WABO 5	368,8,184	392,30,196	910,356,451	5479,233,178	46,22,22	360,3,180	12628,6136,6314
WABO 5_c	96,5,48	116,36,58	254,92,122	786,60,46	64,32,27	96,3,48	5052,2418,2526

than ILP, ETM and TM. A noticeable difference exists between ETM and IMi; ETM took much longer for each log. Second observation is that, not considering FM and TM, no miner has a log on which it performs best on all fitness, precision and generalisation. Tradeoffs have to be made.

IM and ILP did not manage to discover a good 80% model: to achieve perfect fitness, IM sacrifices precision, while ILP sacrifices precision and simplicity. An 80% model was discovered for most logs by HM, but were less simple, not sound, and for some logs discovery took a long time. ETM, with its default settings as tested, focuses on precision and therefore achieves a lower fitness. Moreover, discovery took a long time. IMi discovered sound 80% models quickly in all cases. Regarding precision, two groups of event logs can be identified:

- *BPIC'11 and WABO 1 to WABO 5.* On these logs, IMi produces 80% models with better precision than IM and the baseline FM. Fitness of IMi on all these logs is, as expected for 80% models, higher than ETM, but lower than IM. A manual inspection of the resulting models shows that IMi returns a sequence of activities, whereas IM returns a flower model. Still, some sequential elements are flower models, causing the low precision. Figure 5b shows a part of the model discovered by IMi for WABO 4.
- *BPIC'12 and WABO 1_c to WABO 5_c.* On these logs, IMi discovers good 80% models that can keep up with other miners. Figure 5 shows the results of three miners on the WABO 2_c log. The model discovered by ETM contains the least number of transitions and is obviously the simplest model, but its fitness (0.506) is considerably lower than of IMi (0.946). The main difference is that IMi adds two flower submodels not discovered by ETM, giving a precision of 0.430 for IMi and 0.994 for ETM. For generalisation, both models have the perfect score. Of the 44 activities in WABO 2_c, 23 are not in the model discovered by ETM and only 2 are not in the model discovered by IMi. Therefore, a future trace is more likely to be accepted by the IMi-model than by the ETM-model. Also, note that IMi returned a model in 0.1 seconds and ETM needed 42 minutes, showing that IMi can achieve better results in significantly less time.



(a) IMi on WABO 2_c

(b) IMi on WABO 4 (part of).

(c) ILP on WABO 2_c (part of)

(d) ETM on WABO 2_c

Figure 5: Results of discovery.

5 Conclusion

In this paper, we presented the Inductive Miner - infrequent (IMi), an extension of the Inductive Miner (IM, called B' in [16]) that filters infrequent behaviour locally in each algorithmic step of IM: selecting an operator and a cut, splitting the log and the base cases of the recursion.

Unlike other approaches, IMi can create the so-called 80% model using the Pareto principle while guaranteeing to return a sound process model in a short time. We compared IMi to several existing techniques using performance and soundness, fitness, precision, generalisation and simplicity of the discovered models. IM, HM, ILP and ETM were applied to twelve real-life logs. Compared with IM, models discovered by IMi have a lower fitness, higher precision, equal generalisation and comparable simplicity. IMi always returned a sound 80% model fast, and on all logs scores good on all quality criteria except precision. Results for precision are twofold: on half of the logs, IMi discovered sound 80% models fast, having a lower precision due to discovery of flower models early in the recursion. Note that for many logs, a model scoring well on all quality criteria doesn't exist: process discovery is a tradeoff. On the other half of the logs, IMi discovered better 80% models faster than any other discovery technique, showing the potential of the constructive approach.

Future Work. The parallel operator $\wedge$ remains problematic in operator and cut selection, as none of the features proposed in this paper can filter infrequent behaviour and incompleteness related to this construct. Efficient detection of non-complete parallel logs remains a subject of further research.

References

1. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
2. van der Aalst, W.M.P., et al.: Process mining manifesto. In: Business Process Management Workshops (1). Lecture Notes in Business Information Processing, vol. 99, pp. 169–194. Springer (2011)

3. van Aalst, W.M., van Hee, K.M., van Werf, J.M., Verdonk, M.: Auditing 2.0: Using process mining to support tomorrow's auditor. Computer 43(3), 90–93 (2010)
4. Van der Aalst, W.M., Song, M.: Mining social networks: Uncovering interaction patterns in business processes. In: Business Process Management, LNCS, vol. 3080, pp. 244–260. Springer (2004)
5. van der Aalst, W., de Medeiros, A., Weijters, A.: Genetic process mining. Applications and Theory of Petri Nets 2005 3536, 985–985 (2005)
6. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. Knowledge and Data Engineering, IEEE Transactions on 16(9), 1128–1142 (2004)
7. Adriansyah, A., van Dongen, B., van der Aalst, W.: Conformance checking using cost-based fitness analysis. In: Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International. pp. 55–64. IEEE (2011)
8. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.: Alignment based precision checking. In: Business Process Management Workshops. pp. 137–149. Springer (2013)
9. Buijs, J., van Dongen, B., van der Aalst, W.: A genetic algorithm for discovering process trees. In: Evolutionary Computation (CEC), 2012 IEEE Congress on. pp. 1–8. IEEE (2012)
10. Carmona, J.: Projection approaches to process mining using region-based techniques. Data Mining and Knowledge Discovery 24(1), 218–246 (2012)
11. De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Information Systems 37, 654–676 (2012)
12. van Dongen, B.: BPI Challenge 2011 Dataset (2011), http://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
13. van Dongen, B.: BPI Challenge 2012 Dataset (2012), http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
14. van Dongen, B., de Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: A new era in process mining tool support. Applications and Theory of Petri Nets 2005 3536, 444454 (2005)
15. van Dongen, B.F., Adriansyah, A.: Process mining: fuzzy clustering and performance visualization. In: Business Process Management Workshops. pp. 158–169. Springer (2010)
16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Petri Nets. Lecture Notes in Computer Science, vol. 7927, pp. 311–329. Springer (2013)
17. Mans, R., Schonenberg, M., Song, M., Van der Aalst, W., Bakker, P.: Application of process mining in healthcare–a case study in a dutch hospital. In: Biomedical Engineering Systems and Technologies, pp. 425–438. Springer (2009)
18. de Medeiros, A., Weijters, A., van der Aalst, W.: Genetic process mining: an experimental evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)
19. Smirnov, S., Weidlich, M., Mendling, J.: Business process model abstraction based on synthesis from well-structured behavioral profiles. International Journal of Cooperative Information Systems 21(01), 55–83 (2012)
20. Weijters, A., van der Aalst, W., de Medeiros, A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP 166 (2006)
21. van der Werf, J., van Dongen, B., Hurkens, C., Serebrenik, A.: Process discovery using integer linear programming. Fundamenta Informaticae 94, 387412 (2010)
22. Yzquierdo-Herrera, R., Silverio-Castro, R., Lazo-Cortés, M.: Sub-process discovery: Opportunities for process diagnostics. In: Enterprise Information Systems of the Future, pp. 48–57. Springer (2013)