Interactive Data-driven Process Model Construction

P.M. Dixit[1], H.M.W. Verbeek[1], J.C.A.M. Buijs[1], and W.M.P. van der Aalst[2]

[1] *Eindhoven University of Technology, Eindhoven, The Netherlands*
[2] *Rheinisch-Westflische Technische Hochschule, (RWTH) Aachen, Germany*
{p.m.dixit,h.m.w.verbeek,j.c.a.m.buijs}@tue.nl
wvdaalst@pads.rwth-aachen.de

Abstract. Process discovery algorithms address the problem of learning process models from event logs. Typically, in such settings a user's activity is limited to configuring the parameters of the discovery algorithm, and hence the user expertise/domain knowledge can not be incorporated during traditional process discovery. In a setting where the event logs are noisy, incomplete and/or contain uninteresting activities, the process models discovered by discovery algorithms are often inaccurate and/or incomprehensible. Furthermore, many of these automated techniques can produce unsound models and/or cannot discover duplicate activities, silent activities etc. To overcome such shortcomings, we introduce a new concept to interactively discover a process model, by combining a user's domain knowledge with the information from the event log. The discovered models are always *sound* and can have duplicate activities, silent activities etc. An objective evaluation and a case study shows that the proposed approach can outperform traditional discovery techniques.

Keywords: HCI, process discovery, process mining

1 Introduction

Process discovery, a sub-field of process mining, aims at discovering process models from event logs. Most discovery algorithms aim to do so *automatically* by learning patterns from the event log. Automated process discovery algorithms work well in settings where the event log contains all the necessary (e.g. noise free, complete) information required by the algorithm, and the language of the underlying model is about the same as the language of the models discovered by the discovery algorithm. However, in many real world scenarios this is not the case.

First, the discovered process models might explain the event logs extremely well, but may still be completely incomprehensible to the end user. Therefore, it is imperative to enable the user to have control over the process model being discovered, thereby also enabling incorporation of domain knowledge during process discovery. Second, the process models discovered by discovery algorithms are constrained by the vocabulary of the language used for representing the model, i.e., representational bias [1]. That is, some process discovery algorithms may not discover silent activities (i.e., skippable activities), duplicate activities (i.e., activities that occur more than once) etc. Third, many discovery algorithms may discover process models which are unsound. A sound process model guarantees an option to execute each activity in the process at least once and the ability to reach the final state (thereby terminating the process) from any valid reachable state. In practical settings, unsound process models are often not interesting to the user and hence are discarded. Ideally the discovery algorithm should limit the search space to only sound process models. However, for many discovery techniques this is not the case.

In the interactive process discovery approach presented in this paper and as shown in Figure 1, the user discovers a process model incrementally. The user has total control over the discovery approach and can discover/model the process at a desired complexity level. Therefore, the user can balance simplicity, and to a certain extent, generalization of the process model (among other dimensions) in an efficient way. The interactive editor is further enhanced with process mining capabilities. Information from the event log is used to guide the user in modeling/discovering the process. The user can make informed decisions about where to place a particular activity, depending on the insights gained from the event log. The interactive approach gives the user total control over process discovery. Moreover, by default, the modeled/discovered processes are always in the realm of sound process models.

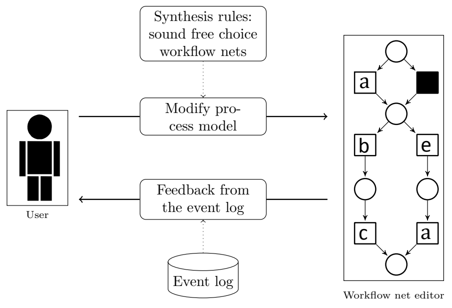

Synthesis rules: sound free choice workflow nets
Modify process model
Feedback from the event log
Event log
User
a
b
e
c
a
Workflow net editor

Fig. 1: Overview of interactive process discovery. The user can edit/discover a sound models guided by an event log.

The remainder of the paper is structured as follows. We review the related work in Section 2 and the preliminaries in Section 3 respectively. In Section 4 we discuss our approach followed by implementation overview in Section 5. In Section 6 we evaluate our approach and conclude in Section 7.

2 Related Work

In this section, we first review the state-of-the-art automated process discovery techniques followed by the user-guided process mining techniques.

2.1 Automated Process Discovery

Discovery algorithms which use the information from the event log about ordering relations in activities and their frequencies such as the α miner and the heuristic miner [3,2] are sensitive to noise. Alongside these, the techniques that use a semantic approach for discovery such as the the state based region miner, numerical abstract domains, and the language based region miner [5,11,6] do not guarantee soundness. Furthermore, algorithms such as the ILP miner and

Table 1: Representational bias of various process discovery and repair algorithms.

	α [2]	HM [3]	ILP [4]	SBR [5]	LBR [6]	ETM [7]	IM [8]	MBR [9]	IPD
Duplicate activities	-	-	-	+	-	+	-	-	+
Silent activities	-	+	-	+	-	+	+	+	+
Self loop activities	-	+	+	+	+	+	+	+	+
Non block structured	+	+	+	+	+	-	-	+	+
Classic soundness*	-	+	-	-	-	+	+	-	+

α : Alpha miner, HM: Heuristic miner, ILP: ILP miner, SBR: State based regions, LBR: Language based regions, ETM: Evolutionary tree miner, IM: Inductive miner, MBR: Model based repair, IPD: Interactive Process Discovery (this paper)
Classic soundness*[10] is defined for the class of workflow nets

the α miner [2,4] cannot discover silent activities. Genetic discovery techniques such as [7] have excessive run times and usually do not give any guarantees on any quality criterion in a short run. The Inductive Miner [8] can discover only block-structured process models and cannot discover duplicate activities. Techniques such as [12] focus on discovering complex process models represented as BPMN models containing sub-processes and events. Discovery techniques such as [13,14,15] discover either constraints based declarative models, or probabilistic models, and not an end-to-end process model.

Our approach differs from all these automated process discovery techniques in multiple ways (see Table 1 for an overview). The process models generated by our approach are always sound, since we use a synthesis rules kit (based on [16]) which guarantees soundness. In our approach, the user has control over the discovery (modeling) process, therefore addition of constructs such as duplicate activities, silent activities, (self-)loops etc. are all allowed as deemed appropriate by the user. Also, noisy (incomplete) information could be ignored (overcome) based on the extracted information from the event log presented to an informed user.

2.2 User-guided Approaches

Of late, there has been an interest in using domain knowledge, along with the event logs, for process discovery [17,18,19,20]. However, the domain knowledge is usually represented by some pre-defined constructs or some sort of ‘rules’ which are used as input during the process discovery. The language used to represent the domain knowledge severely limits the expressiveness of the domain expert. In our approach, the user has total control over the discovery phase and can intuitively use the domain knowledge, along with the event logs, to interactively visualize and discovery the process model. In [21], the authors provide a way for the user to include domain expertise in the α miner. However, this approach is tied to the underlying α algorithm, and thereby includes all the limitations of the α algorithm. AProMore [22] provides a host of techniques including interactive process model repair techniques [23]. However, the focus of these techniques is model repair using alignments, similar to [9]. These techniques have a different scope compared to our approach, whose focus is process model discovery.

3 Preliminaries

In this section, we introduce the so-called *activity logs* and *synthesized nets*.

3.1 Activity Logs

An *activity log* is a multi-set (or bag) of sequences of activities. Every sequence of activities in the activity log is called an activity trace. Table 2 shows an example activity log, which contains 20 activity traces, 7 different activity traces, 87 activities, and 6 different activities. In real life, we often see *event logs* instead of activity logs, where an event log is a (multi-)set of sequences of *events*. An event is typically a key-value pair containing values for several attributes that are related to the event, such as the time on which the event occurred and to which activity the event relates. Using the latter attribute, the mapping from such an event log to an activity log is then straightforward. As a result, although we use activity logs in this paper, the presented approach is also applicable on event logs.

Table 2: The example activity log

Activity trace	**Freq**
$\langle a, a, f, c\rangle$	8
$\langle a, f, a, e, c\rangle$	1
$\langle a, f, a, e, e, c\rangle$	1
$\langle b, d, e, c\rangle$	4
$\langle b, d, e, e, c\rangle$	1
$\langle b, e, d, c\rangle$	4
$\langle a, a, f, e, e, e, c\rangle$	1

3.2 Synthesized nets

A *synthesized net* is a free-choice [16] workflow net [24] containing a source place i, a sink place o, a start transition $\top$, and an end transition $\perp$. Only the start transition may have an input arc from the source place, and only the end transition may have an arc to the sink place. The source and sink place are known from the workflow nets [24], but our approach also requires the additional start and end transition.

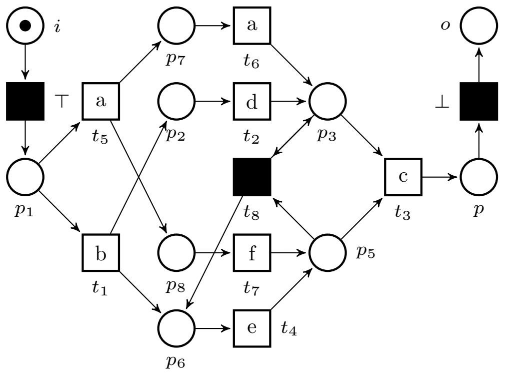


Fig. 2: The example synthesized net corresponding to Table 2.

We use the regular Petri net semantics [16] for the synthesized net. A transition is enabled if all its input places contain tokens. In the initial state of the example net, as shown in Figure 2, only the $\top$ transition is enabled as only place i contains a token. An enabled transition may be fired, which removes a token from every input place and adds a token to every output place. Firing the enabled transition $\top$ in the example net results in the state where place p_1 contains one token and all other places are empty. This state enables the transitions t_1 and t_5, etc. Being a workflow net, the goal of the example net is to reach the state where only place o contains a token and all other places are empty, that is, the stop state of the net. The soundness property [10] guarantees that this stop state can always be reached.

Figure 2 also shows that we allow the transitions to be labeled (by activities, actually), but that we also allow a transition to be silent, that is, not labeled. As examples, the transition t_5 is labeled with activity a, and the transition t_8 is silent. The start and end transitions are required to be silent.

4 Approach

In this Section, we discuss the approach used in order to enable interactive process discovery using the synthesis rules [16]. We begin with the initial net and the synthesis rules in the context of our approach.

4.1 Initial net and synthesis rules

Figure 3 shows the initial synthesized net, which is the minimal synthesized net. As mentioned earlier, the places i and o, the transitions $\top$ and $\bot$, and the arcs connecting them are required, and we need the place p with its connecting arcs to make it a workflow net. We have three synthesis rules, where each of the rules is based on a synthesis rule from [16]: an abstraction rule, a place rule, and a transition rule.

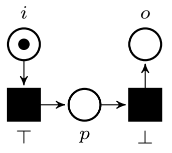
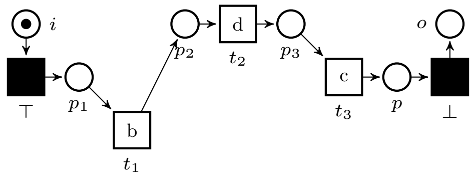


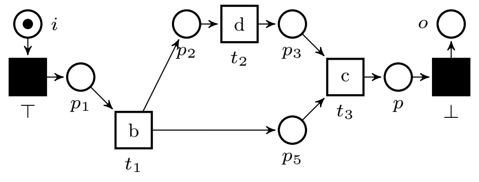
Fig. 3: Initial net.

The abstraction rule allows one to introduce a new place and a new transition, provided that we have a set of transitions R and a set of places S such that there is an arc from every transition in R to every place in S. Figure 4a shows a synthesized net that can be obtained from the initial net by applying this rule three times. First, place p_1 and transition t_1 are introduced using $R = \{\top\}$ and $S = \{p\}$. Second, place p_2 and transition t_2 are introduced using $R = \{t_1\}$ and $S = \{p\}$. Third, place p_3 and transition t_3 are introduced using $R = \{t_2\}$ and $S = \{p\}$.

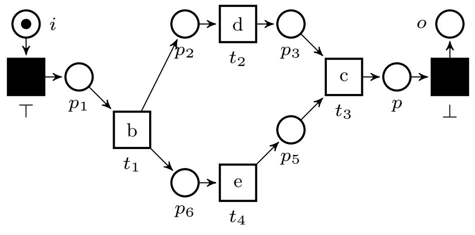
The other two synthesis rules use the fact whether a place or a transition is *linearly dependent*. For this, we need the *incidence matrix* of the net. A net can be short-circuited by removing the place o, and re-routing the outgoing arc from $\bot$ to i. Table 3 shows the incidence matrix of the short circuited version of Figure 4b. In this matrix, every row vector corresponds to a place, and every column vector corresponds to a transition. The number at the intersection of a row vector and a column vector provides the net effect of firing that transition on that place. For e.g., the value for place p_1 and transition $\top$ is 1, as firing $\top$ adds a token to p_1.



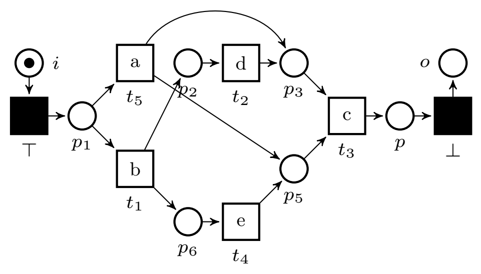
(a) A part of the example synthesized net.



(b) After having added the place p_5 using the place rule.



(c) After having applied the abstracton rule for the fifth time.



(d) After having added the transition t_5 using the transition rule.

Fig. 4: First steps in synthesizing the example synthesized net using the synthesis rules.

Table 3: Incidence matrix of 4b.

	$\top$	t_1	t_2	t_3	$\bot$
i	-1	0	0	0	1
p_1	1	-1	0	0	0
p_2	0	1	-1	0	0
p_3	0	0	1	-1	0
p	0	0	0	1	-1
p_5	0	1	0	-1	0

The place rule allows one to introduce a new place, provided that the new place is linearly dependent [16] on the existing places, from the incidence matrix. Figure 4b shows a synthesized net that can be obtained from the net in Figure 4a by using a single application of this rule: the place p_5, which is linearly dependent on the places p_2 and p_3, has been introduced. In a similar way, the transition rule allows one to introduce a new transition, provided that the new transition is linearly dependent [16] on the existing transitions. Figure 4d shows a synthesized net obtained from Figure 4c using a single application of this rule: the transition t_5, which is linearly dependent on the transitions t_1, t_2, and t_4, has been introduced. We can deduce the net of Figure 2 from Figure 4d by using two more abstraction rules and one more transition rule.

In [16] it has been proven that these rules preserve well-formedness [16], which is related to soundness but not exactly the same. The key difference is that for soundness we require a specific state (one token in the place i) whereas for well-formedness we do not require such a state. Hence, in our approach the place rule is restricted to forbid adding places which may preserve well-formedness but may not preserve soundness. The related intricacies are out of scope.

4.2 Using Activity Logs

In this Section, we discuss how the information from the activity logs is linked to the synthesized nets and presented to the user. The activity logs are central for making decisions in all the automated process discovery techniques. In our case, the information from the activity logs is extracted and presented to the user for decision making by projecting it on the synthesized net, along with normalized values in a tabular form. The synthesized net expands one transition and/or one place at a time. The user labels newly added transitions in the synthesized net with either an activity from the activity log, or the transition does not represent an activity, i.e. leaves the transition silent. The information from the activity log is aggregated in a pairwise manner between each activity from the synthesized net (labels of visible transitions) and the activity selected by the user to be added to the net, which assists the user in positioning the selected activity in the net. We first define the co-occurs value between two activities.

Definition 1 (Co-occurs ($C_{(a,b)}$)). *Let L be an activity log and let a and b be two activities that occur in L, the co-occurs value of (a,b), denoted $C_{(a,b)}$ is:*

$$C_{(a,b)} = \frac{|[\sigma \in L \mid a \in \sigma \wedge b \in \sigma]|}{|[\sigma \in L \mid a \in \sigma]|}$$

For a pair of activities a, b, if the co-occurs value is 0, then a and b do not occur together, whereas if the co-occurs value is 1 then b occurs if a occurs. Next, we define the eventually follows value, which indicates the number of times an activity is eventually followed by another activity.

Definition 2 (Eventually follows). *Let L be an activity log and let a and b be two activities that occur in L, and for a trace $\sigma \in L$, let $\#a >_\sigma b$ indicate number of occurrences of b after the first occurrence of a in σ. The eventually follows relationship between a and b for a trace σ, denoted $EF_\sigma(a,b)$, is:*

$$EF_\sigma(a,b) = \begin{cases} \dfrac{\#a >_\sigma b}{\#a >_\sigma b + \#b >_\sigma a} & , \textit{if } \#a >_\sigma b \neq 0 \\ 0 & , \textit{otherwise.} \end{cases}$$

The eventually follows value for a pair (a,b) w.r.t. the entire log L, denoted as $EF_L(a,b)$, is calculated as the average of all the traces which have a non-zero value for a pair, or is zero otherwise. If $EF_L(a,b) = 1$, then a and b co-occur, but a never occurs after a b, which hints that b should be following a. The *directly follows* relation w.r.t. activities a and b is calculated as follows:

Definition 3 (Directly follows ($DF_{(a,b)}$)). *Let L be an activity log and let a and b be two activities that occur in L, and for a trace $\sigma \in L$, let $\#a >^d_\sigma b$ indicate number of occurrences of b directly after the occurrence of activity a in σ. The directly follows relationship between a and b for a trace σ, denoted $DF_\sigma(a,b)$, is:*

$$DF_\sigma(a,b) = \begin{cases} \dfrac{\#a >^d_\sigma b}{\#a >^d_\sigma b + \#b >^d_\sigma a} & , \textit{if } \#a >^d_\sigma b \neq 0 \\ 0 & , \textit{otherwise.} \end{cases}$$

The directly follows value for the entire log L, denoted as $DF_L(a,b)$, is calculated similar to $EF_L(a,b)$. Similarly, *eventually precedes* and *directly precedes* values are also calculated. The sum of eventually (directly) follows and precedes values for a pair is either 1 or 0.

4.3 Activity Log Projection

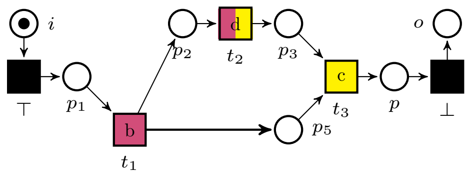


Fig. 5: Eventually follows/precedes projections on Figure 4c, when the selected activity is e and the pairwise relations between activities are: $EF_L(b, e) = 1, EF_L(e, d) = 0.5, EF_L(e, c) = 1, C_{(e,d)} = 1, C_{(e,b)} = 1$ and $C_{(e,c)} = 1$.

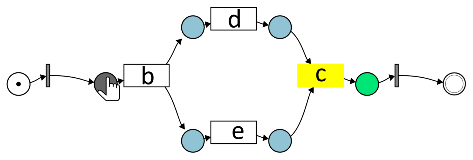
Here, we describe how the information from the activity log is visualized. The information from the activity log is projected on the synthesized net and also presented as tables. As a first step, the user selects an activity from all the activities available in the activity log. The user can choose to either project the information from the activity logs either based on eventually follows (precedes) relation, or directly follows (precedes) relation. Depending on the activity selected by the user, the coloring of the activities in the current synthesized net is updated. The colors indicate which activities (and to what extent) from the synthesized net occur before and/or after the selected activity. The opacity of the colors indicate the co-occurrence of the two activities.

For example, Figure 5 shows the projection on transitions when an activity e is selected by the user. The degree of *purple (yellow)* color in a transition indicates that the selected activity occurs *after (before)* the activity represented by the transition. As transition t_1 is completely colored purple (darker), we know that activity e occurs after activity b. Likewise, as transition t_3 is completely colored yellow (lighter), we know that activity e occurs before activity c. In contrast, as transition t_2 is fifty-fifty colored purple and yellow we know that activity e occurs about equally often before and after activity d. The opacity of the coloring indicates the co-occurrence values of the activity chosen and the activities represented by the transitions. Based on these insights, it is clear that activity e must be added in parallel to d, before c and after b, i.e., using the abstraction rule on the thicker edge in Figure 5. Multiple transitions having the same label would also have the same coloring. If a transition is colored white, it implies that the activity selected and the activity represented by the transition never co-occur together. Furthermore, the user is also presented with raw information pertaining to the selected activity such as *% of the traces in which the selected activity occurred*, *average occurrence of the selected activity in a trace* etc.

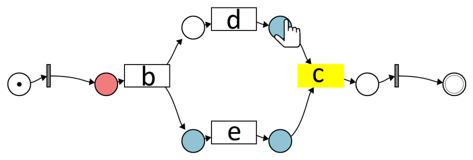
5 Interactive Editing and Implementation

In this section, we discuss the implementation and user interaction details of our technique. The proposed technique is implemented and available in the "Interactive Process Mining" package in the nightly build version of ProM[3]. The user

[3] http://www.processmining.org/prom/start



(a) Possible outputs when the place before b is selected. Dark grey colored place indicates possibility of a *self-loop*, i.e. a transition having same input and output. Blue colored places indicate candidate transition with multiple outputs, for the selected input place. Each green colored place indicate candidate with a single output place for the selected input place(s).



(b) After selecting the place after d as an output place. The input place (i.e. the place before b) is turned red, indicating it is no longer possible to use this place to add a *self-loop* transition. The green places are filtered out, as such candidate transitions are no longer valid for the selected input-output place combination. Moreover, the place before d is also colored white as there is no candidate transition which contains both the place after d and the place before d in its output places.

Fig. 6: User interaction for adding linearly dependent transitions in our tool.

interacts with a synthesized net (starting with the initial net), in order to deduce new nets, by applying one of the synthesis rules. In order to use the abstraction rule, the user clicks on a (set of) arc(s), and presses *enter*. The selected arcs are highlighted in green. The abstraction rule allows addition of a new place and a new transition in between a set of transitions and a set of places. The (optional) activity label of the new transition is pre-selected by the user, after which the rule is applied.

In the case of linear dependency rules, all possible applications of a rule are projected on the synthesized net based on the user interaction. We explain this with the help of an example from Figure 6, which shows a screen shot of our tool (corresponding to Figure 4c) and an application of the transition rule, when activity a is selected by the user. Note that activities b, d and e are colored white and activity c is colored yellow. Hence we would like to place activity a as an alternative to b, d and e and before c. Re-collect that the transition rule allows addition of a linearly dependent transition to a synthesized net. All the candidate linearly dependent transitions are pre-computed. This set of candidate transitions is finite, and independent of the activity label chosen by the user. Whenever a user navigates on a place, all the candidate transitions are explored and only those candidate transitions are presented which have the navigated place as an input place. For all such candidate transitions, the output places (and possible additional input places) are projected on the synthesized net, using color coding. The user first selects the desired input places of the candidate transition. The candidate transitions are filtered based on the selected input places, and the output places of the filtered candidate transitions are highlighted. Next, the user chooses output places. In case of multiple output places for a candidate transition, the user clicks on one of the desired multiple output place. The candidate transitions are further filtered based on the output places chosen. When a user has selected enough input and output places to pinpoint a single candidate transition, the selected candidate transition is added to the synthesized net. A similar approach is followed for the place rule.

6 Evaluation

In order to validate the approach presented in this paper, we presented two types of evaluations: an objective comparison with traditional process discovery approaches, and a subjective validation via a case study.

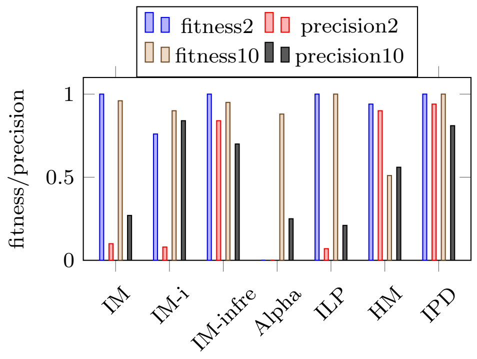
6.1 Objective Evaluation : Process Discovery Contest

As a part of the objective evaluation of the tool, we use our winning entry from the annual process discovery contest[4], organized at the BPI workshop of BPM 2017 conference. The aim of the process discovery contest is to evaluate tools and techniques which are able to discover process models from incomplete, infrequent event logs. In total, 10 process models had to be discovered from 10 corresponding event logs. Every model could contain sequences, choices and concurrent activities. Furthermore, the organizers provided participants with additional information about each process model, such as the presence of duplicate activities, loops, optional activities, inclusive choices and long-term dependencies. We consider this additional information as the *domain knowledge* in our approach. Every final process model was evaluated based on an unseen test log containing 20 traces, as well as by a jury of BPM practitioners to rate the simplicity and understandability of the discovered process models.

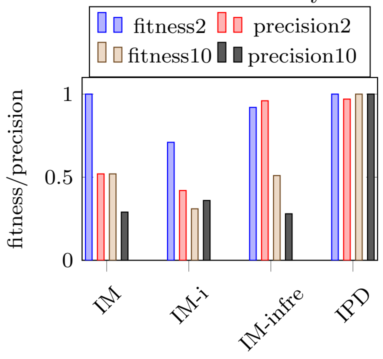
In order to keep the comparison across various discovery techniques fair, in this section we focus on 2 process models: model numbers 2 and 10. Other process models (1 and 3-9) contained artifacts such as inclusive choices. Even though our approach is able to model such artifacts, many state-of-the-art techniques can not. Both the training logs for models 2 and 10 are incomplete logs, and contain sequential, concurrent, exclusive choices and optional activities. Furthermore, model 2 contains loops and model 10 contains duplicate activities. We used this information in combination with simple log visualizers, in order to discover process models with our tool. Both the process models discovered were able to replicate all the behavior from the test activity logs. Furthermore, over all the 10 models, we received 98.5% accuracy w.r.t. the test activity logs: of the 200 test traces we had to classify, we classified 197 correctly.

In Figure 7a we compare the fitness and precision scores of the training activity log used to discover the process model, with the discovered process model, using a prefix-based alignment technique [25]. For model 2, the α miner resulted in a model which could not be replayed using [25]. Furthermore, in Figure 7b we compare the discovered process models with the original process models which are used to generate the activity logs (both training and test logs). This comparison is done using the technique from [26], which supports only the process models discovered by inductive miner variants and our approach. The fitness value indicates the part of the behavior of the activity log (or the original process model) that is captured by the discovered process model. The precision value on the other hand captures the part of behavior of the discovered

[4] https://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_discovery_contest



(a) Discovered models compared with training activity log using [25].



(b) Discovered models compared with the original models used to generate the activity logs of the process discovery contest, using [26].

Fig. 7: Discovery techniques - IM : Inductive Miner, IM-i : Inductive Miner incomplete, IM-infreq : Inductive Miner infrequent, Alpha miner, ILP Miner, HM: Heuristic Miner, IPD : Interactive Process Discovery - this paper.

model that is also present in the activity log (or the original process model). As evident from Figure 7a and Figure 7b, our approach typically outperforms all the automated discovery techniques. By using the properties of the net (such as possibility of loops, duplicates etc.) we were able to interactively discover process models which are strikingly similar to the original process models. For example, Figure 8 shows the synthesized net we discovered from activity log 10. The only real difference when compared to the original process model is the incorrect presence of the silent transition just below the transition labeled f.

6.2 Case study

In this section, a real-life synthesized net is modeled using our tool by using domain knowledge along with the event log. The case study is performed with a local healthcare partner who was the domain expert, by using data on the treatment process of 2 years for a specific type of cancer. The domain expert knew parts of the process behavior and was interested in discovering the end-to-end process

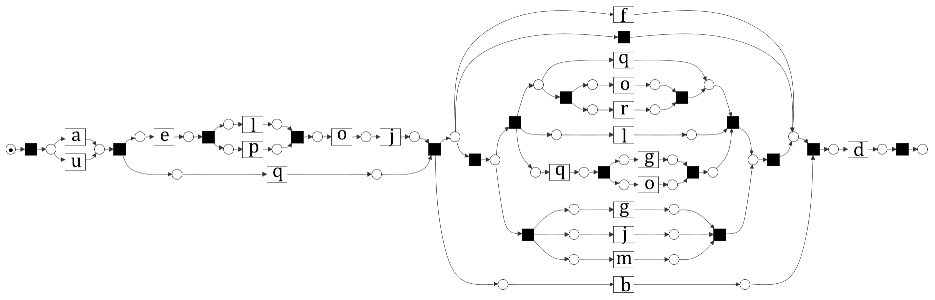


Fig. 8: Synthesized net 10 of the process discovery contest discovered using our approach.

model. Although in theory the process should be rather straightforward, the usage of automated process discovery techniques resulted in extremely incomprehensible process models, or highly imprecise process models which allowed for any behavior. The process models discovered by the traditional discovery techniques were either incomprehensible and/or very far off from the reality according to the domain expert. Therefore, the interactive process discovery approach was used to try to structure the process data by using domain knowledge.

The synthesized net discovered using our approach is shown in Figure 9. The domain expert had complete control about the modeling of the process. On several occasions, the domain expert took assistance from insights of the activity log gained via our technique. For e.g., the domain expert was not entirely sure if *LvPA* should be placed before or after *MDO*. However, after gaining insights from the data, the domain expert decided to add *LvPA* before *MDO*. On some other occasions, the domain expert chose to ignore the information from the data, deeming it inappropriate and/or inadequate. Finally, using the information from the event log and some background knowledge, the domain expert was able to discover, and was satisfied with, a very structured process model.

The interactively discovered process model had a low fitness with the activity log. Investigating the causes for the low fitness led to the conclusion that there were some serious data quality issues in the activity log, rather than non-compliance of protocols. Data quality problems are often the reason why all the automated discovery algorithms fail to discover a structured process model. However in our case, by not relying completely on an event log for process discovery, and using knowledge from a domain expert, a structured process model was discovered. Moreover, since the domain expert was directly involved in process modeling/discovery, the simiplicity and generalization dimensions of the process model were implicitly taken into account.

7 Conclusion and Future Research

In this paper, we presented the concept of interactive discovery of a process model based on synthesis rules. The use of synthesis rules as an engine to expand

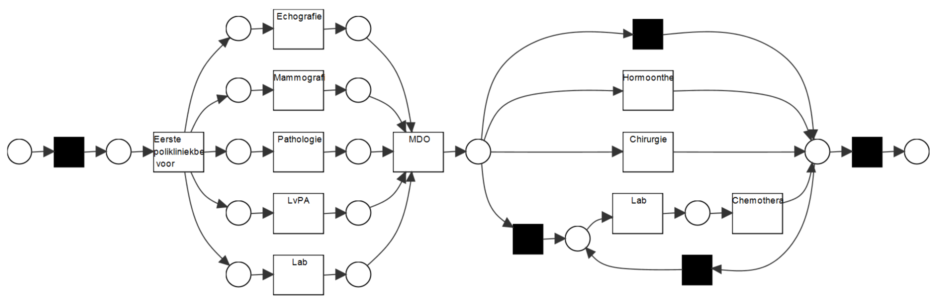


Fig. 9: Synthesized net for cancer patients in a Dutch hospital as discovered by the domain expert using our tool.

the synthesized net guarantee soundness of the discovered process model. Furthermore, the information from the event log is extracted and presented to the user to assist the user in decision making. Giving users complete control over the discovery approach supported by the information from the event log enables critical decision making. This is true especially when all the information needed by the discovery algorithms is not present in the event log; which is often the case in many real-life event logs as apparent from the case study presented. The automated discovery algorithms fail to cope with insufficient information in the event log, and could produce process models which are incomprehensible and/or inaccurate. Moreover, our approach is able to discover constructs such as duplicate activities, inclusive choices and silent activities, that cannot be discovered by many state-of-the-art techniques. In the future, we aim to improve the assistance provided to the user in decision making. One future direction would be to provide online conformance results to the user during process discovery. Another future direction could be pre-populating a number of options to be presented to the user to add a particular activity.

References

1. van der Aalst, W.M.P.: On the representational bias in process mining. In: Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on. (June 2011) 2–7
2. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: Discovering process models from event logs. Knowledge and Data Engineering, IEEE Transactions on **16**(9) (2004) 1128–1142
3. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. Integr. Comput.-Aided Eng. **10**(2) (April 2003) 151–162
4. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: International Conference on Applications and Theory of Petri Nets, Springer (2008) 368–387
5. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE transactions on computers **47**(8) (1998) 859–882
6. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: International Conference on Business Process Management, Springer (2007) 375–383
7. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: Evolutionary Computation (CEC), 2012 IEEE Congress on, IEEE (2012) 1–8
8. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Business Process Management Workshops, Springer (2014) 66–78
9. Fahland, D., van der Aalst, W.M.P. In: Repairing Process Models to Reflect Reality. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 229–245
10. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing **23**(3) (May 2011) 333–363

11. Carmona, J., Cortadella, J.: Process discovery algorithms using numerical abstract domains. IEEE Transactions on Knowledge and Data Engineering **26**(12) (Dec 2014) 3064–3076
12. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Beyond tasks and gateways: Discovering bpmn models with subprocesses, boundary events and activity markers. In Sadiq, S., Soffer, P., Völzer, H., eds.: Business Process Management, Cham, Springer International Publishing (2014) 101–117
13. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems. In: Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. Springer Berlin Heidelberg, Berlin, Heidelberg (2009) 278–295
14. Bellodi, E., Riguzzi, F., Lamma, E.: Probabilistic declarative process mining. In Bi, Y., Williams, M.A., eds.: Knowledge Science, Engineering and Management, Berlin, Heidelberg, Springer Berlin Heidelberg (2010) 292–303
15. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Quarterly **40**(4) (2016)
16. Desel, J., Esparza, J.: Free choice Petri nets. Volume 40. Cambridge university press (2005)
17. Greco, G., Guzzo, A., Lupa, F., Luigi, P.: Process discovery under precedence constraints. ACM Transactions on Knowledge Discovery from Data **9**(4) (June 2015) 32:1–32:39
18. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on, IEEE (2011) 192–199
19. Dixit, P.M., Buijs, J.C.A.M., van der Aalst, W.M.P., Hompes, B.F.A., Buurman, J. In: Using Domain Knowledge to Enhance Process Mining Results. Springer International Publishing, Cham (2017) 76–104
20. Rembert, A.J., Omokpo, A., Mazzoleni, P., Goodwin, R.T.: Process discovery using prior knowledge. In: Service-Oriented Computing. Springer (2013) 328–342
21. Mathern, B., Mille, A., Bellet, T.: An Interactive Method to Discover a Petri Net Model of an Activity. working paper or preprint (April 2010)
22. La Rosa, M., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: Apromore: An advanced process model repository. Expert Systems with Applications **38**(6) (2011) 7029 – 7040
23. Armas Cervantes, A., van Beest, N.R.T.P., La Rosa, M., Dumas, M., García-Bañuelos, L.: Interactive and incremental business process model repair. In Panetto, H., Debruyne, C., Gaaloul, W., Papazoglou, M., Paschke, A., Ardagna, C.A., Meersman, R., eds.: On the Move to Meaningful Internet Systems. OTM 2017 Conferences, Cham, Springer International Publishing (2017) 53–74
24. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of circuits, systems, and computers **8**(01) (1998) 21–66
25. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards robust conformance checking. In: Business Process Management Workshops. Volume 66 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2011) 122–133
26. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Software & Systems Modeling **08**(Jul) (2016) 1619–1374