An Interactive Method to Discover a Petri Net Model of an Activity

Benoît Mathern[1,2], Alain Mille[1], and Thierry Bellet[2]

[1] Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69621, France
http://liris.cnrs.fr/
[2] INRETS, LESCOT, 25 avenue François Mitterrand, 69675 Bron cedex, France
http://www.inrets.fr/

Abstract. This paper focuses on *interactive Knowledge Discovery processes* in the context of understanding an activity from behavioural data. Data mining provides patterns experts have to interpret and synthesize as new knowledge. Discovering patterns is an analysis task while building new symbolic knowledge is a synthesis task. A previous trace based approach (ABSTRACT) offered a first answer to support analysis. This paper goes one step forward in supporting the synthesis task. We modify an algorithm of automata discovery in order to involve the user in the mining process, exploiting his expert knowledge about the observed activity. We chose the α-algorithm (Van Der Aalst et al.) developed for Petri nets discovery in a workflow management context. The modified algorithm is described and illustrated, showing how to use intermediate data to converge interactively to a satisfying automata. Finally, we discuss the use of this approach to contribute to a new knowledge mining process.

Key words: Knowledge discovery, Petri nets, Interactive mining

1 Introduction

1.1 General Context of the Work: Man Machine Interactive Knowledge Discovery

The general context of this work is what we could call "*man machine interactive Knowledge Discovery*" which can be considered as a special case of knowledge engineering approach. Actually, if the knowledge engineering approach finds its roots in the general question of knowledge acquisition, representation and maintenance in order to develop knowledge based systems (KBS) [7,12], its scope becomes larger and larger for supporting construction of ontologies in knowledge management context [8] and in the semantic web context [14]. A number of tools have been developed to face this problem of semantic integration of data, information and even people [17]: methods and tools for sharing ontologies; algorithms, heuristics and machine learning for ontologies mapping; and, because of the lack of clearly identified experts, machine learning techniques to

mine available data, texts and logs [18]. Choosing the corpus of data, texts or events to mine is an expert task. For example, the machine learning process has to be tuned to produce relevant ontologies which are checked by experts. A first issue is to get relevant data, texts and events among what is available in the environment and to prepare these sources to properly build ontologies. A second issue is to support the Knowledge Discovery process leading to create new knowledge about the domain. A third issue is to take into account what is discovered to guide new Knowledge Discovery. Beyond discovering relevant concepts for a process, it becomes increasingly important to discover the dynamics of these processes. Discovering knowledge by observing processes, through their behaviour and their productions, is not new and there is a strong tradition in the Human Computer Interactions research [19] to propose concrete methods for discovering explicit knowledge from various observation sources (computer events, video and audio records, annotations, data, texts, etc.). We can use the general notion of "interaction traces" for referring to such knowledge sources and we are specifically interested in exploiting traces of events for supporting Knowledge Discovery of the dynamics of processes.

1.2 Modeled Trace and Trace Based System

Considering such interaction traces as "first class" computer objects, our research team developed the notions of *modeled traces ($\mathcal{M}$-Traces)* and of *trace based system (TBS)* [13]. A modeled trace *$\mathcal{M}$-Trace* is constituted of both the sequence of temporally situated *observed elements (obsels)* which constitutes the instantiated trace and of the model of this trace which gives the semantics of the observed elements and of the relations between them. A *trace based system (TBS)* is a framework managing such $\mathcal{M}$-Traces and providing trace oriented services such as: collecting a $\mathcal{M}$-Trace, computing sequences satisfying a pattern in a specific $\mathcal{M}$-Trace, transforming a $\mathcal{M}$-Trace into another one for abstraction purpose (filtering, merging, reformulating obsels), navigating through the transformation graph of a $\mathcal{M}$-Trace... In the context of this paper, we assume these services are available and we take advantage of such a TBS which is embedded in the Abstract software. In this framework, traces are represented in RDF[3] and their models are managed with the *Protégé* tool[4]. Using such a TBS allows to *prepare* and *transform* sequences of events as $\mathcal{M}$-Traces at the relevant level of abstraction with explicit semantics. The Abstract framework [10] has been developed in order to support the Knowledge Discovery process in the context of driver behaviour analysis [9]. In this context, a huge quantity of data is collected during driving sequences, from numerous sensors, video capture, observer annotations... Then, the analyst uses the Abstract interface to progressively transform traces in order to discover relevant patterns according to some hypothesis on the driver behaviour[5].

[3] http://www.w3.org/RDF/
[4] http://protege.stanford.edu/
[5] For details and Abstract demonstration see http://liris.cnrs.fr/abstract/

1.3 Towards an Interactive Knowledge Discovery Process

Fayyad et al. [6] proposed a general process for discovering knowledge from data (see figure 1 on the following page). We propose a slightly different approach when data are collected from the observation of some process, using a Trace Based System that provides both services of preparation and transformation to support analysis. Furthermore, and following the general idea defended by R. Michalski [16] we propose to complete this Knowledge Discovery process by a computer supported synthesis step providing a formal knowledge representation (for simulation purpose for example). This proposition is sketched in figure 2 on the next page. These knowledge oriented approaches offer opportunities to provide feedback controls at each step with knowledge formal representations and easy interactions with analysts (experts) to understand results and to control the full Knowledge Discovery process.

1.4 AUTOMATA as a Support for Interpretation of the Dynamics of Processes

The ABSTRACT approach allows to consider the request to find a pattern as the *signature* of a new concept which can then be added to the ontology that describes activity traces. This is a first answer to the general issue of considering "knowledge mining" instead of "data mining" according to R. Michalski [16] since this process allows to enrich the formal representation of the activity. In order to go a step further we propose a process, AUTOMATA[6], to synthesize the knowledge about the dynamics of the observed process, extending the Knowledge Discovery process by supporting the interpretation task (automata description) with the help of a trace mining process focusing on the available occurences of patterns satisfying a particular request. In order to build such an assistance, we propose to modify algorithms able to synthesize automata from traces. In this paper, we focus on an algorithm developed by Aalst et al. [4] in the context of workflows management. These algorithms assume that the automata traces are complete which, of course, is not often the case when observing a real process. In order to overcome this problem, we propose an *interactive* version of this algorithm to allow the analyst to mobilize his own knowledge to compensate for the lake of information in the process traces.

The state of the art (section 2) presents different data mining approaches for automata discovery, explains the choice of Petri Nets discovery and describes the α-algorithm chosen to illustrate the *man machine interactive Knowledge Discovery principle*. The contribution is detailed in section 3 and is illustrated with a simple example. The section 4 discusses the first results and future contributions: different ways the modified algorithm can guide the analyst to complete intermediate results in order to get a relevant automata by combining observed events and known elements about the observed activity.

[6] AUTOMATA means: AUTOmata Modelling of the Activity, based on Trace Analysis.

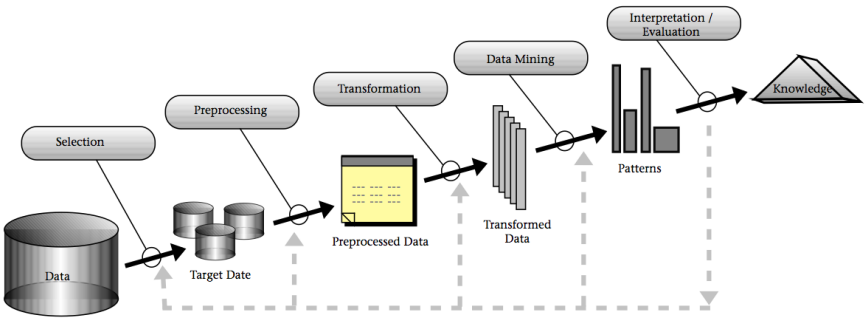

Fig. 1. In this process, all steps are supposed to be automatic but the last one, *interpretation*, is the only one which allows to tune the previous steps.

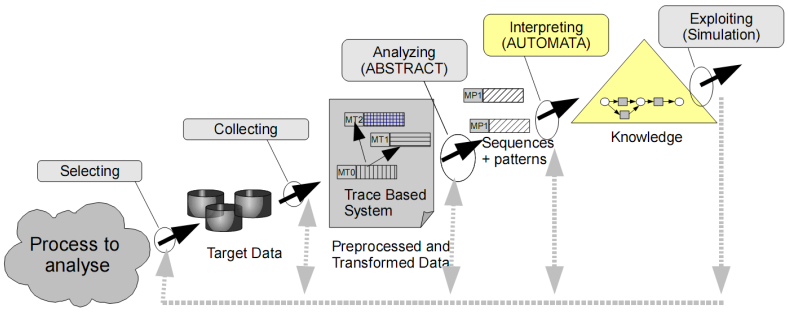

Fig. 2. In our approach, the user can interactively mine traces to produce relevant patterns (ABSTRACT step). Discovered knowledge (ontology) allows to tune the collecting and the analysing steps. In this paper, we introduce an explicit support (AUTOMATA) to the "synthesis" step, opening a new step "*Exploiting*" which completes the process. In our context, the formal knowledge representation of dynamics is a Petri Network, allowing further exploitation (for simulation for example).

2 State of the Art

2.1 Overview of Approaches for Building an Automata from Traces

Automata Approaches for Knowledge Representation An automaton has a graphical representation that can be understood by a non-specialist, and at the same time, it is an *operative* representation of knowledge. Both those qualities are important for our approach as we want both the analyst to understand the discovered knowledge and the discovered knowledge to be used for producing an activity.

Actually, automata are used both for producing a behaviour (simulation) and for describing (specification) expected behaviours (for example, the Statechart formalism or UML state diagrams).

The literature proposes several ways to discover an automaton that could produce a given set of traces. We focused on the *Process Mining* approaches [3,2]. Those approaches deal with logs coming typically from an entreprise's workflow management system. In this context a prescriptive model of the workflow exists (how people should do). However, it does not guarantee that people are following this prescriptive workflow. The goal is to discover a model of the real workflow, as people actually instantiate it, in order to compare this actual workflow to the prescriptive workflow and see how the workflow can be optimised.

Workflow Mining Issues Van der Aalst and Weijters [3] made a review of the issues of Process Mining. There are several challenging issues that cannot be tackled together by any single algorithm:

- mining hidden tasks (a task that does not appear in the traces),
- mining duplicate tasks (a task that would correspond to several states or transitions of a model),
- mining loops,
- mining different perspectives (from the same traces, focusing on different aspects of the activity),
- dealing with noise,
- dealing with incompleteness,
- visualizing results,
- ...

Van der Aalst et al. [2] introduce four process mining algorithms: EMiT [1] and Little Thumb [5,21] (both based on the α-algorithm), InWoLvE [11] and Process Miner [20].

The authors compare the abilities of these four algorithms to discover known models (to *rediscover* a model). We have selected four properties of the algorithm from their comparison: the ability to deal with time, parallelism, loops and noise.

Table 1 on the following page presents a comparison of these algorithms according to these four properties. This summary shows that EMiT approach is interesting on many aspects but does not deal with noise. Little Thumb and

InWoLvE both deal with noise, but not with time. Process Miner neither deal with time nor with noise. In the context we are studying (analysing human activity), it is important to be able to deal with parallelism and with loops, which they all can do.

Table 1. Comparison of four process mining approaches, adapted from Van der Aalst et al. [2].

Approach	Time	Basic Parallelism	Loops	Noise
EMiT	Yes	Yes	++	No
Little Thumb	No	Yes	++	Yes
InWoLvE	No	Yes	+	Yes
Process Miner	No	Yes	+	No

In our context of use (human activity analysis and synthesis), the ability to represent loops and parallelism is important. The ability to deal with time (EMiT) or noise (Little Thumb or InWoLvE) would be a nice benefit. As both EMiT and Little Thumb are based on a same algorithm, the α-algorithm, we propose to make this common algorithm interactive. This would make it possible for an analyst to deal with the incompleteness issue of the data: the analyst can provide the algorithm with additional knowledge.

The α-algorithm is our first target to add interactivity to a mining algorithm. It does not support itself all the types of loops, but, the resulting algorithm can be integrated into EMiT or Little Thumb.

2.2 The α-algorithm: Definitions and Basic Presentation

The formal definitions introduced here are taken from Van der Aalst et al. [4]. For a complete formal definition of the notions used here and the proof of the algorithm, please refer to Van der Aalst et al. [4].

Given (1) a Petri-net with some properties, and (2) a collection of traces of this model, with some properties, the α algorithm can reproduce the Petri-net (1) from the collection of traces (2).

We explain now more precisely what type of Petri-nets we are considering; then, what type of collections of traces we need; and finally, how the α algorithm works.

Petri-Nets Van der Aalst et al. use a variant of Petri-net called Place/Transition nets.

Definition 1 (P/T-nets). *A Place/Transition net, or simply P/T-net, is a tuple* (P, T, F) *where:*

1. P *is a finite set of* places,
2. T *is a finite set of* transitions *such that* $P \cap T = \emptyset$, *and*

3. $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs, called the* flow relation.

The authors define a *workflow net* (WF-net) as a particular subset of P/T-nets. The figure 3 presents an example of WF-net. WF-nets have an input place $i \in P$, with no flow relation directed to i, an output place $o \in P$, with no flow relation starting from o and by linking the output place o to the input place i with a new transition (t), the resulting P/T-net $((P, T', F') = (P, T \cup \{t\}, F \cup \{(o,t),(t,i)\}))$ is strongly connected, that is to say for any two places or transitions $x, y \in P \cup T'$, there is a path of flow relations in F' from x to y.

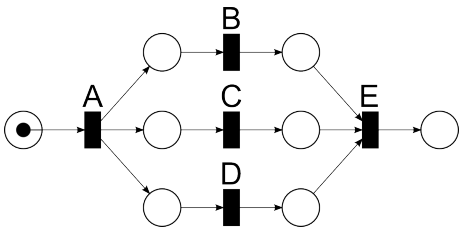

Fig. 3. A WF-net with 3 parallel transitions. Circles are places, black rectangles with labels are transitions and arcs are flow relations. The black dot represents a token on the input place.

The authors define the notion of soundness of a WF-net. In a few words, a WF-net $N = (P, T, F)$ is *sound* if,

- it is *safe*: while firing transitions, starting from the initial marking (only the input place has a token), there is no more than one token on each place.
- While firing transitions (starting from the initial marking), if there is a token on the output place (final marking), then there is no other token on any other place of the WF-net.
- It is possible to reach this final marking (from the initial marking).
- Any transitions of the WF-net is reachable from the initial marking.

The authors define a *structured workflow net* (SWF-net) as a particular type of WF-net, with some structural properties: no implicit place, a choice between two possible activities or a synchronisation between two activities is possible with some constraints illustrated in figure 4 on the following page.

Workflow Logs The authors define the notions of workflow trace and workflow log. Given a set of tasks T, a *workflow trace*, noted σ is a sequence of tasks of T ($\sigma \in T^*$). A *workflow log*, noted W, is a set of workflow traces.

Transitions of a WF-net correspond to tasks of a workflow. Thus, the workflow trace of a WF-net represents the sequence of execution of the different transitions of this WF-net.

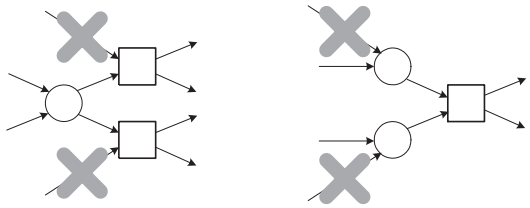

Fig. 4. Two constructs not allowed in SWF-nets [4].

Van der Aalst et al. [4] define several relations that can be deduced from WF-logs W. We will introduce three of these. For a WF-log W of a WF-net $N = (P, T, F)$, and for $a, b \in T$:

1. $a >_W b$ means there is a trace $\sigma \in W$ where b directly follows a;
2. $a \rightarrow_W b$ iff $a >_W b$ and $b \not>_W a$, which means there is a trace in W where b directly follows a, and in no trace of W a directly follows b; and
3. $a \#_W b$ iff $a \not>_W b$ and $b \not>_W a$, which means in no trace of W b directly follows a and in no trace of W a directly follows b.

In order to use the previous relations, the authors define the notion of complete workflow log. Given a sound WF-net N, a workflow log W of N is complete if (1) "all tasks that potentially directly follow each other in fact directly follow each other in some trace in the log" (2) each transition of N appears at least once, in at least one trace σ of the workflow log W.

α-algorithm Previous notions allow us to introduce the α-algorithm [4]. We explain its meaning step by step.

Definition 2 (Mining algorithm α). *Let W be a workflow log over T. $\alpha(W)$ is defined as follows.*

1. $T_W = \{t \in T \mid \exists_{\sigma \in W}\ t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in W}\ t = \mathit{first}(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in W}\ t = \mathit{last}(\sigma)\}$,
4. $X_W = \{(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B}\ a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A}\ a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B}\ b_1 \#_W b_2\}$,
5. $Y_W = \{(A, B) \in X_W \mid \forall_{(A', B') \in X_W}\ A \subseteq A' \wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\}$,
6. $P_W = \{p_{(A,B)} \mid (A, B) \in Y_W\} \cup \{i_w, o_w\}$,
7. $F_W = \{(a, p_{(A,B)}) \mid (A, B) \in Y_W \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_W \wedge b \in B\} \cup \{(i_w, t) \mid t \in T_I\} \cup \{(t, o_w) \mid t \in T_O\}$, *and*
8. $\alpha(W) = (P_W, T_W, F_W)$.

The authors prove that given a complete WF-log W of a sound SWF-net $N = (P, T, F)$ with no short loops, the α-algorithm can reconstruct N from W, modulo the name of the places. We explain its meaning in the following.

1. *T_W is the set of events observed in the WF-log traces. It corresponds to the set of transitions of the reconstructed WF-net.*
2. *T_I is the set of input transitions of the reconstructed WF-net.*
3. *T_O is the set of output transitions of the reconstructed WF-net.*
4. *X_W characterises a set of candidate places of the reconstructed WF-net. It is a set of couples (A, B) where A and B are sets of transitions, such as any transition of A is the origin of a flow relation to this candidate place and this candidate place is the origin of flow relations to any transition of B. The sets (A, B) are constructed such as for any two elements $a_1, a_2 \in A$, there is no trace where a_1 is directly followed by a_2 and no trace where a_2 is directly followed by a_1 (the same for any two elements in B), and for any couple of elements $(a, b) \in (A, B)$, there is a trace where a is directly followed by b and no trace where b is directly followed by a. The figure 5 on the next page illustrates this property.*
5. *Y_W characterises the places of the reconstructed WF-net (without the input and output places). Y_W it the set of maximum elements of X_W. If a candidate place $(A, B) \in X_W$ is such as there exists another candidate place $(A', B') \in X_W$ with $A \subseteq A'$ and $B \subseteq B'$, then either $(A, B) = (A'B')$, and the candidate place is kept, either the place (A, B) is redundant and does not contain more information than the candidate place (A', B'). The algorithm does not keep such implicit places.*
6. *P_W is the set of places of the reconstructed WF-net. Each couple (A, B) of Y_W is explicitly defined as a place $p_{(A,B)}$, and the input (i_w) and output (o_w) places are added.*
7. *F_W is the set of flow relations between places in T_W and transitions in P_W. The connection between places is defined as for each place $p_{(A,B)}$ each transition of A is connected by a flow relation to the place $p_{(A,B)}$, and the place $p_{(A,B)}$ is connected by a flow relation to each place of B. In addition, F_W also contains the flow relation between the input place (i_w) and the input transitions (T_I) and between the output transitions (T_O) and the output place (o_w)*
8. *$\alpha(W)$ is the reconstructed WF-net.*

3 Contribution: an Interactive Variant of the α-algorithm

In this section, we describe how we have adapted the α-algorithm to make it interactive. We also describe some further tracks to make interactions stronger and smarter.

As several extensions of this algorithm exist (for example the α^{++}-algorithm [15]), we keep the possibility to go further without loosing the interactive property.

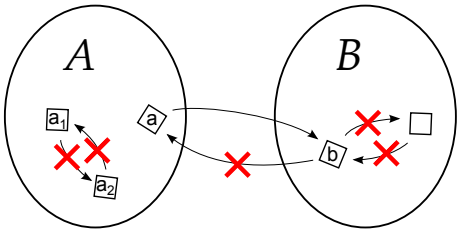

Fig. 5. An illustration of elements (A, B) of the X_W set at the step 4 of the α-algorithm. A and B are two sets of transitions. An arrow between two transitions illustrates that there is a trace in the WF-log where the first transition is directly followed by the other ($>_W$ relation). A crossed out arrow between to transition means there is no trace in the WF-log where the first transition is directly followed by the other ($\not>_W$ relation).

3.1 From the α-algorithm to the α^i-algorithm

We want to make the α-algorithm interactive. This way an expert, a human who wants to discover knowledge, can exploit his or her knowledge to help the α-algorithm to (re-)discover a workflow net.

Why is it necessary for an expert to introduce some of his or her knowledge in the mining step? Because in most cases, it is not possible to know whether the collection of traces (WF-log) we mine is complete or not. Even worse, sometimes, especially when dealing with data related to human activity, we know the collection of traces is incomplete, noisy, and even missing data related to not directly observed[7] behavior.

We consider three possibilities for a human to interact with the α-algorithm.

- Before applying the algorithm: by *preparing* traces, by *adding traces* as examples, or by *giving negative examples* (a behaviour that the constructed model should not produce).
- On the results of the algorithm: by interpreting the output model, by *renaming places* of the P/T-net, by *editing* the places and flow relations, *adding* non-observable transitions.
- In the meanwhile: without changing the algorithm, by *editing* some intermediary results of the α-algorithm, or by changing the algorithm, implementing a mechanism of mixed chaining that asks the user for (possibly) missing information.

In order to keep the good properties of the α-algorithm, allowing extensions such as EMiT or Little Thumb, we focus on the edition of intermediate results of the α-algorithm. Obviously, each step of the α-algorithm is an intermediary result. We have chosen to focus on one particular intermediary result which is

[7] A behaviour may be observable on a video for a human expert, but not actually logged in a computer-readable trace

not explicitly defined in the algorithm: the definition of the $>_W$ relation. Indeed, step 4 uses the $\rightarrow_W$ and $\#_W$ relations that are defined from the $>_W$ relation. We propose to present the $>_W$ relation to the analyst, so that he or she can edit it, adding his or her own knowledge to the knowledge implicitly contained in the traces.

This particular $>_W$ relation seems interesting for the analyst because it can be easily translated in natural langage and thus understandable to the analyst. Actually, $a >_W b$ stands for "there is a trace where b directly follows a". But in the case of incomplete traces (WF-logs), the question should be slightly different, as we know everything is not observable in the traces. The real question of our concern is "is it possible that a trace where b directly follows a exists". The answer to this question would lead to the definition of a new relation, that we note $>_\mathcal{A}$ ($\mathcal{A}$ stands for "Analyst"). From this new relation, we can rely on the knowledge of the analyst to make the α-algorithm useful on incomplete traces.

Then, by using $>_\mathcal{A}$ relation, relations $\rightarrow_\mathcal{A}$ and $\#_\mathcal{A}$ are straightforwardly defined as follows. For all $a, b \in T_W$:

- $a \rightarrow_\mathcal{A} b$ if and only if $a >_\mathcal{A} b$ and $b \not>_\mathcal{A} a$, and
- $a \#_\mathcal{A} b$ if and only if $a \not>_\mathcal{A} b$ and $b \not>_\mathcal{A} a$.

On this basis, we propose the first modification of the α-algorithm as follows, that we will refer as the α^i-algorithm.

- Steps 1 to 3 are the same:
  1. $T_W = \{t \in T \mid \exists_{\sigma \in W}\ t \in \sigma\}$,
  2. $T_I = \{t \in T \mid \exists_{\sigma \in W}\ t = \mathit{first}(\sigma)\}$,
  3. $T_O = \{t \in T \mid \exists_{\sigma \in W}\ t = \mathit{last}(\sigma)\}$,
- Between step 3 and 4, we introduce 3 additional steps, to prompt the analyst for additional knowledge:
  (a) build the $>_W$ relation from traces,
  (b) display the definition of $>_W$ in natural langage and propose the analyst to complete this relation with his or her knowledge (resulting in the $>_\mathcal{A}$ relation), and
  (c) from the $>_\mathcal{A}$ relation, build the $\rightarrow_\mathcal{A}$ and $\#_\mathcal{A}$ relations.
- Then, steps 4 to 8 are exactly the same, except from that they implicitly use knowledge coming from expertise ($\mathcal{A}$), not only from WF-logs or traces (W). In order to formally show this difference, we rewrite those steps here:
  4. $X_\mathcal{A} = \{(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B}\ a \rightarrow_\mathcal{A} b \wedge \forall_{a_1, a_2 \in A}\ a_1 \#_\mathcal{A} a_2 \wedge \forall_{b_1, b_2 \in B}\ b_1 \#_\mathcal{A} b_2\}$,
  5. $Y_\mathcal{A} = \{(A, B) \in X_\mathcal{A} \mid \forall_{(A', B') \in X_\mathcal{A}}\ A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B')\}$,
  6. $P_\mathcal{A} = \{p_{(A,B)} \mid (A, B) \in Y_\mathcal{A}\} \cup \{i_w, o_w\}$,
  7. $F_\mathcal{A} = \{(a, p_{(A,B)}) \mid (A, B) \in Y_\mathcal{A} \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_\mathcal{A} \wedge b \in B\} \cup \{(i_w, t) \mid t \in T_I\} \cup \{(t, o_w) \mid t \in T_O\}$, and
  8. $\alpha^i_\mathcal{A}(W) = (P_\mathcal{A}, T_W, F_\mathcal{A})$.

3.2 Illustrative Example

We propose to show the interest of our approach on a simple example.

For this example, we will work with the P/T-net model presented in figure 3 on page 7 (that is a sound SWF-net matching the properties needed for the α-algorithm). From this P/T-net model, we produce the following incomplete set of traces (WF-log): $ABCDE$, $ACBDE$, $ABDCE$.

As expected from the α-algorithm, using this incomplete WF-log produce random results. In this particular case, it produces a meaningful P/T-net, but not the one we are looking for (figure 6).

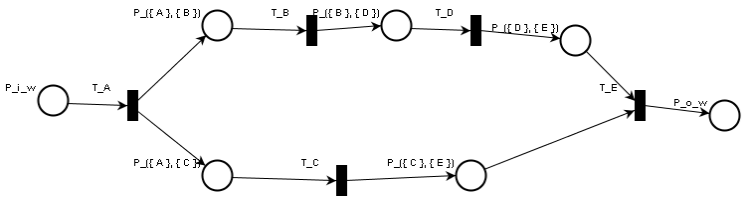

Fig. 6. A resulting P/T-net of the α-algorithm on incomplete traces (WF-log).

With some knowledge coming from the analyst (figure 7), the algorithm is able to rediscover the original P/T-net (figure 8 on the next page).

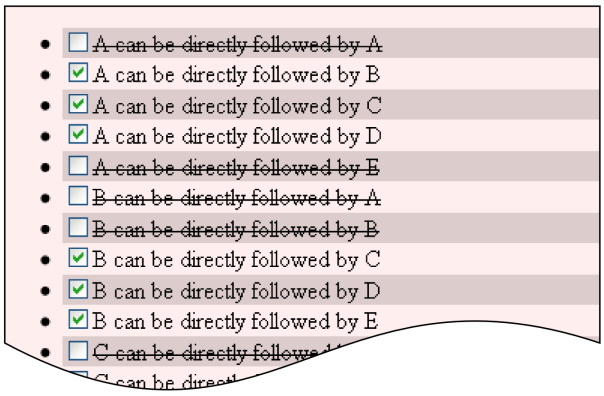

Fig. 7. A screenshoot presenting the interface the expert can interact with (building the $>_{\mathcal{A}}$ relation).

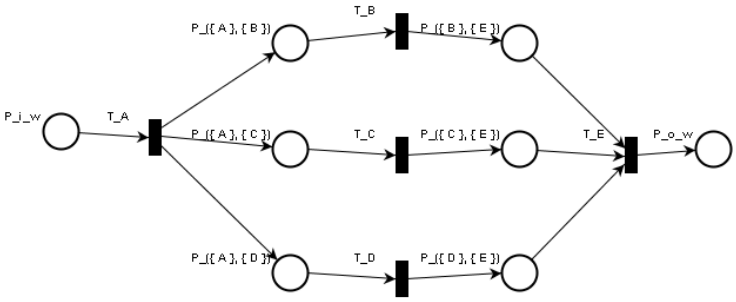

Fig. 8. A resulting P/T-net of the interactive α-algorithmfrom an incomplete traces (WF-log) completed by analyst's knowledge.

4 Discussion and Perspectives

4.1 Discussion

We have shown that it is possible to adapt an algorithm, the α-algorithm to make it interactive. The user, in our case an expert in the workflow, can interact with the algorithm in order to add knowledge that was not available in the traces. This interaction can be a base for dealing with the incompleteness issue of the traces we collect.

The interaction we propose is based on the $>_W$ relation, that might be incomplete (or erroneous, due to noise in the data). We propose that the user, based on the computed $>_W$ enrich it to produce the $>_A$ relation. However, this step hides a complexity problem. Listing all the possible $>_A$ relations for a WF-net with n transition is $O(n^2)$. For the example shown in figure 7 on the facing page, the WF-net studied has 5 transitions, the user has to check the 25 possible relations.

This problem would appear nearly hopeless with a traditional mining approach. However, in the context of interactive Knowledge Discovery, introduced at the very beginning of this article, we think we can master this issue. The knowledge discovery is guided by a user, who knows what type of knowledge he or she is looking for. From this top-down approach, it is possible, at the first steps of the process to select data, analyse it with ABSTRACT framework, before using α^i-algorithm.

4.2 Perspectives

The first immediate improvement for the α^i-algorithm, would be to decline it to the improved version of the α-algorithm, like in EMiT or Little Thumb. We already started to work on making the α^{++}-algorithm interactive. The α^{++}-algorithm [15] is a variant of the α-algorithm dealing with short loops.

It would also be interesting to enrich the possibilities of interaction. As seen in section 3.1, the user could interact at three levels: before applying the algorithm (on the traces), after applying the algorithm (on the resulting model), or in the meanwhile (interacting with intermediary results or with a mechanism of mixed chaining).

In the future, we will investigate more specifically the possibility (1) to develop a mixed chaining, and (2) to interact with traces.

Mixed chaining would make it possible to reduce the effort of the expert to use an algorithm by suggesting to him or her possible missing information. For example, the algorithm could propose to the user "almost discovered" places or flow relations. This tool would, we believe, improve the usability of such an interactive algorithm.

Interacting with traces corresponds to integrating the interactive mining algorithm into a software platform that guarantees that it is easy to loop back and forth from knowledge discovered, to traces, to mining algorithms (figure 2 on page 4). Thus, we integrate the interactive version of the algorithm, in a software platform capable of interacting with the ABSTRACT platform. This integration has to come along with a methodology to discover knowledge, that we will develop and instantiate on car-driving behavioural data, to support the cognitive modelling of the driving activity.

5 Conclusion

This paper proposes a first approach to mine an observed activity process in order to interactively construct descriptive and operational knowledge. We show it is possible to adapt a workflow mining algorithm, the α-algorithm, to achieve this interactive discovery approach. Human knowledge and intermediate mining results are interactively woven to construct an automata that can be used as an overall knowledge representation and as an operational knowledge representation available for computer exploitation (simulation). Therefore, such an interactive algorithm supports the general idea of making the overall knowledge discovery process interactive (figure 2 on page 4).

Acknowledgments. This work was supported by Région Rhône-Alpes and the "Transport, Territories and Society" Cluster.

References

1. Aalst, W., Dongen, B.: Discovering Workflow Performance Models from Timed Logs. In: Proceedings of the First International Conference on Engineering and Deployment of Cooperative Information Systems. pp. 45–63. Springer-Verlag (2002)
2. Van der Aalst, W., Van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47(2), 237–267 (2003)

3. Van der Aalst, W., Weijters, A.: Process mining: a research agenda. Computers in Industry 53(3), 231–244 (2004)
4. van der Aalst, W., Weijters, A., Maruster, L.: Workflow mining: Which processes can be rediscovered? In: Eindhoven University of Technology. pp. 1–25 (2002)
5. van der Aalst, W., Weijters, T.: X-tra-KLeinduimpje in Workflowland: Op zoek naar procesdata. Scope 10(12), 38–40 (2002)
6. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery: an overview. AI-Magazine pp. 37–54 (1996)
7. Feigenbaum, E.A., McCorduck, P.: The fifth generation. Addison Wesley (1983)
8. Fensel, D.: Ontology-based knowledge management. Computer 35, 56–59 (2002)
9. Georgeon, O., Henning, M.J., Bellet, T., Mille, A.: Creating cognitive models from activity analysis: A knowledge engineering approach to car driver modeling. In: International Conference on Cognitive Modeling. pp. 43–48. Taylor & Francis (July 2007)
10. Georgeon, O., Mille, A., Bellet, T.: Analyzing behavioral data for refining cognitive models of operator. Philosophies and Methodologies for Knowledge Discovery pp. 588–592 (2006)
11. Herbst, J.: Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen. Ph.D. thesis, Universitt Ulm (2001)
12. Kendal, S.L., Creen, M.: An introduction to knowledge engineering. Springer, London (2007)
13. Laflaquière, J., Settouti, L.S., Prié, Y., Mille, A.: A trace-based system framework for experience management and engineering. In: Second International Workshop on Experience Management and Engineering (EME 2006) in conjunction with KES2006. vol. 4253, pp. 1171–1178. Springer, Berlin (2006)
14. Lee, T.B.: The semantic web. Scientific American (2001)
15. de Medeiros, A., van Dongen, B., van der Aalst, W., Weijters, A.: Process mining: Extending the α-algorithm to mine short loops. Eindhoven University of Technology, Eindhoven (2004)
16. Michalski, R.S.: Knowledge mining: A proposed new direction. Invited talk at the Sanken Symposium on Data Mining and Semantic Web, Osaka University, Japan, March 10-11, 2003 (march 2003)
17. Noy, N.F.: Semantic integration: a survey of ontology-based approaches. ACM Sigmod Record 33(4), 65–70 (2004)
18. Omelayenko, B.: Learning of ontologies for the web: the analysis of existent approaches. In: In Proceedings of the International Workshop on Web Dynamics (2001)
19. Sanderson, P.M., Fisher, C.: Exploratory sequential data analysis: Foundations. Human-Computer Interaction 9, 215–317 (1994)
20. Schimm, G.: Process Miner-A Tool for Mining Process Schemes from Event-Based Data. In: Proceedings of the European Conference on Logics in Artificial Intelligence. pp. 525–528. Springer-Verlag (2002)
21. Weijters, A., Van Der Aalst, W.: Workflow mining: discovering workflow models from event-based data. In: Dousson, C., Höppner, F., Quiniou, R. (eds.) Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data. pp. 78–84 (2002)