Journal of Intelligent Information Systems (2019) 53:93–112
https://doi.org/10.1007/s10844-019-00543-2

Activity prediction in process mining using the WoMan framework

Stefano Ferilli[1,2] · **Sergio Angelastro**[1]

Received: 15 January 2018 / Revised: 18 December 2018 / Accepted: 2 January 2019 /
Published online: 17 January 2019

Abstract
Process Management techniques are useful in domains where the availability of a (formal) process model can be leveraged to monitor, supervise, and control a production process. While their classical application is in the business and industrial fields, other domains may profitably exploit Process Management techniques. Some of these domains (e.g., people's behavior, General Game Playing) are much more flexible and variable than classical ones, and, thus, raise the problem of predicting which activities will be carried out next, a problem that is not so compelling in classical fields. When the process model is learned automatically from examples of process executions, which is the task of Process Mining, the prediction performance may also provide indirect indications on the correctness and reliability of the learned model. This paper proposes and compares two strategies for activity prediction using the WoMan framework for workflow management. The former proved to be able to handle complex processes, the latter is based on the classic and consolidated Naïve Bayes approach. An experimental validation allows us to draw considerations on the pros and cons of each, used both in isolation and in combination.

Keywords Process mining · Activity prediction · Process model

1 Introduction

Process Management techniques are useful in domains where a production process must be monitored (e.g. in the industry) in order to check whether the actual behavior is compliant with a desired one. When a process model is available, new process enactments can be automatically supervised. The complexity of some domains requires to learn automatically the process models, because building them manually would be very complex, costly, and error-prone. Process Mining (Weijters and van der Aalst 2001; IEEE Task Force on Process Mining 2012) approaches aim at solving this problem.

✉ Stefano Ferilli
stefano.ferilli@uniba.it

[1] Dipartimento di Informatica, Università di Bari, Bari, Italy

[2] Centro Interdipartimentale per la Logica e sue Applicazioni, Università di Bari, Bari, Italy

A relevant issue in Process Management in general, and in Process Mining in particular, is to assess how well can a model provide hints about what is going on in the process execution, and what will happen next. Indeed, given an intermediate status of a process execution, knowing how the execution will proceed might allow the (human or automatic) supervisor to take suitable actions that facilitate the next activities. This task, known as *activity prediction*, may be stated as follows: given a process model and the current (partial) status of a new process execution, guess which will be the next activity that will take place in the execution. In industrial environments, the rules that determine how the process must be carried out are quite strict; so, predicting the process evolutions is a trivial consequence of conformance checking. Other, less traditional application domains (e.g., the daily routines of people at home or at work, seen as a process), involve much more variability, and obtaining reliable predictions becomes both more difficult and more useful. Another, very important and interesting, application of process-related predictions is in the assessment of the quality of a model. Indeed, since models are learned automatically exactly because the correct model is not available, only an empirical validation can be run. In literature, this is typically done by applying the learned model to new process enactments.

Given these motivations, we focused on the activity prediction task and developed two approaches to carry it out. Both are based on the WoMan framework for workflow management (Ferilli and Esposito 2013; Ferilli 2014), which proved able to outperform other state-of-the-art systems when dealing with very complex processes. The former (the *heuristic* approach) is the one originally adopted by WoMan (Ferilli et al. 2016). The latter (the *Bayesian* approach) is a novel contribution of this paper: based on the Naïve Bayes algorithm, it was developed in order to check whether and how this classical and consolidated Machine Learning technique can improve activity prediction performance. Other original contributions of this paper are the following: first, the WoMan's process model representation is extended by introducing additional information aimed at improving the prediction performance; second, the detailed prediction algorithms (especially the Bayesian one) are reported for the first time; third, an implementation of the two approaches in WoMan is used to compare their performance on several real-world datasets.

This paper is organized as follows. Section 2 reports some basics on process mining and related works. The next two sections present WoMan, its (extended) formalism and its approach to activity prediction. Then, Section 5 reports the details of the novel Bayesian approach. Section 6 reports and comments about the experimental outcomes. Finally, in the last section, some conclusions are drawn and future work issues outlined.

2 Basics & related work

A *process* consists of actions performed by agents (Agrawal et al. 1998; Cook and Wolf 1996). A *workflow* or *process model* is a formal specification of a process. It may involve sequential, parallel, conditional, or iterative execution (van der Aalst 1998). A process execution, compliant to a given workflow, is called a *case*. It can be described as a sequence of *events* (i.e., identifiable, instantaneous actions, including decisions upon the next activity to be performed), associated to *steps* (time points) and collected in *traces* (van der Aalst et al. 2004). Relevant events are the start and end of process executions, or of activities (Cook and Wolf 1996). Traces may, in turn, be collected in *logs*. A *task* is a generic piece of work, defined to be executed for many cases of the same type. An *activity* is the actual execution of a task by a *resource* (an agent that can carry it out).

Process models (i.e., workflows) are usually expressed as some kind of graph, where vertexes are associated to tasks and edges represent causal connections among tasks. While several specialized formalisms have been proposed in the literature, Petri Nets are a classical and widespread representation used for process models. In particular, Workflow Nets (van der Aalst 1998) are a restriction of Petri Nets purposely defined for workflow models, that allows to express XOR and AND splits/joins, representing, respectively, alternative and concurrent executions. *Declarative* process mining approaches (Pesic and van der Aalst 2006) learn models expressed as a set of constraints, rather than a monolithic model.

Various process mining algorithms have been proposed in literature for the discovery of process models. Cook and Wolf (1996) proposed an inference method which combines a statistical and an algorithmic approach to discover process models with a trade-off between accuracy and robustness to noise. Greco et al. (2005) proposed a hierarchical clustering-based method to partition the event log and discover a process model for each element of the partition, after which all the models are merged in a single one. A method that finds the best Hidden Markov Model (HMM) that reflects the process model was proposed by Herbst and Karagiannis (1998). van der Aalst et al. (2004) introduced the α-algorithm that generates Workflow Nets based on the relations found in the event log. Since the α-algorithm can not handle all constructs, an extended version that overcomes this problem has been proposed by Wen et al. (2006). van der Aalst et al. (2005) exploited genetic algorithms to build Petri nets from causal matrixes. This approach tackles problems such as noise, incomplete data, non-free-choice constructs, hidden activities, concurrency, and duplicate activities. However, the need to set parameters and long runtimes are significant drawbacks of this technique.

Specifically, activity prediction, classified as an *operational support* task, received very little attention in literature so far. Schonenberg et al. (2008) cast it as a recommendation problem. The process model is used to provide a ranked list of the currently enabled activities, given the partial execution of the enacted process and historical information. The experimental results they report show that the more historical information used, the better the quality of the recommendation. A limitation of their approach is that it considers traces that are simple sequences of activities. A Bayesian approach was used by Cook and Wolf (1996) to guess the next activity in a process case, as a function of its frequency. Starting from a Markov approach used to infer a formal model, the authors move towards a Bayesian problem to infer activities. However, this approach was not thoroughly investigated. Other approaches naturally fit very well-structured process, but suffer from problems related to incompleteness, noise, underfitting and overfitting. The approach considered in Ceci et al. (2014) overcomes these problems using a customized sequential pattern mining algorithm. Unfortunately, it can only handle sequences, and concurrent behavior is not considered.

The WoMan framework (Ferilli and Esposito 2013; Ferilli 2014) lies at the intersection between Declarative Process Mining and Inductive Logic Programming (ILP) (Muggleton 1991). Indeed, it pervasively uses First-Order Logic as a representation formalism, that provides a great expressiveness potential and allows one to describe contextual information using relationships. Experiments proved that WoMan can handle efficiently and effectively very complex processes, involving:

- a very large number of tasks,
- high concurrency (i.e., tasks whose execution occurs simultaneously),
- duplicated tasks (i.e., multiple tasks with the same label),
- hidden tasks (i.e., tasks that exist in the model but not in its event log),
- short loops (i.e., loops involving just one or two tasks),
- and nested loops,

thanks to its powerful representation formalism and process handling operators. Differently from all previous approaches in the literature, it is *fully incremental*: not only can it refine an existing model according to new cases whenever they become available, it can even start learning from an empty model and a single case, while others need a (large) number of cases to draw significant statistics before learning starts. This allows to carry out continuous adaptation of the learned model to the actual practice efficiently, effectively and transparently to the users (Ferilli 2014). In particular, very good performance was obtained by WoMan on the activity prediction task (Ferilli et al. 2017a) (such as in predicting the next chess move, outperforming other state-of-the-art techniques proposed and discussed in Lai (2015) and Oshri and Khandwala (2016)) for which reason we took it as the basis for the research presented in this paper.

3 The WoMan formalism

WoMan representations are based on the Logic Programming formalism (Lloyd 1987), and specifically on Datalog (Ceri et al. 1990), where only constants or variables are allowed as terms. Following foundational literature (Agrawal et al. 1998; Herbst and Karagiannis 1999), trace elements in WoMan are n-tuples, represented in WoMan as facts

$$\texttt{entry(}T\texttt{,}\ E\texttt{,}\ W\texttt{,}\ P\texttt{,}\ A\texttt{,}\ O\ \texttt{[,}\ R\texttt{])}\ \texttt{.}$$

that report information about relevant events for the case they refer to.

T is the event timestamp;[1]
E is the type of the event, whose allowed values are:

- **begin_process**,
- **begin_activity**,
- **end_activity**,
- **end_process**,
- **context_description**;

W is the name of the workflow the process refers to;
P is a unique identifier for each process execution;
A is:

- **start** (a reserved fictitious activity), when $E =$ **begin_process**,
- the name of the activity that is being initiated, when $E =$ **begin_activity**,
- the name of the activity that is being terminated, when $E =$ **end_activity**,
- **stop** (a reserved fictitious activity), when $E =$ **end_process**,
- a description of contextual information at time T, in the form of a conjunction of FOL atoms built on domain-specific predicates, when $E =$ **context_description**;

O is the progressive number of occurrence of that activity in that process execution;
R (an optional field) can be used to specify the agent that carries out activity A.

Activity begin and end events are needed to properly handle time span and parallelism of tasks (van der Aalst et al. 2004). Since parallelism among activities is explicit, there is no

[1] It can be of any representation of time points (e.g., milliseconds from process case start, date-time in YYYYMMDDHHMMSS format, progressive integers, etc).

need for inferring it by means of statistical (possibly wrong) considerations. In each case, the activities are uniquely identified by a progressive number called *step*.

Given a set of training cases $\mathcal{C}$, WoMan learns a model consisting of a set of atoms built on several predicates, each expressing a different kind of constraint (Ferilli et al. 2017b). The core of the model, established in its very first version, is expressed by predicates[2] `task/2` and `transition/4`.

- `task`(t, C_t) : task $t \in \mathcal{T}$ (where $\mathcal{T}$ is the set of possible tasks for the process under consideration) occurred in training cases $C_t \subseteq \mathcal{C}$.
- `transition`(I, O, t, C_t) : transition[3] t, occurred in training cases $C_t \subseteq \mathcal{C}$, is enabled if all input tasks $I = [i_1, \ldots, i_n]$ are running; if fired, after stopping the execution of all tasks in I (in any order), the execution of all output tasks $O = [o_1, \ldots, o_m]$ is started (again, in any order). If several instances of a task can be running at the same time, I and O are multisets, and application of a transition consists in closing as many instances of running tasks as specified in I and starting as many executions of new tasks as specified in O. A transition is completed when the execution of all of its output tasks terminates.

`task/2` atoms express the tasks that are allowed in the process. `transition/4` atoms express the allowed connections between activities in a very modular way. A convenient notation for expressing transitions is

$$t : I \Rightarrow O \; [C_t]$$

where the C_t parameter can be omitted if irrelevant. Let us denote by $\mathcal{P}$ the set of transitions for the process under consideration.

Compared to classical representations, in which the overall topology of the graph is fixed, this representation breaks the process models in several small pieces, that might in principle be recombined together in many ways. The recombination occurs when the input multisets of different transitions have non-empty intersection with the output multisets of other transitions. This mechanism can lead to non-determinism or unexpected behaviors, never seen before. These issues are tackled by extending the model with suitable predicates (reported below) that constrain the allowed combinations during the supervision phase as specified in Section 4.1. To enforce irredundancy, WoMan exploits a number of additional information items. A fundamental one is the C_t parameter. First, and most important, it allows WoMan to check that all transitions involved in a new execution were all involved in the same (at least one) training case (Ferilli 2014). Second, it allows WoMan to compute the probability of a task or transition t, as the relative frequency $|S_t|/n$ where $n = |\mathcal{C}|$ is the number of training cases and S_t is the set of different elements in C_t. This can be used for process simulation, for activity prediction and for noise handling (ignoring all tasks/transition in the model whose probability does not pass a specified noise threshold). Third, it allows WoMan to bound the number of repetitions of loops. Indeed, C_t is a multiset, because if a task or transition t was executed k times in case c, then C_t includes k occurrences of c. So, WoMan knows the maximum number of times that a task or transition can be executed in the same case.

Transitions can be seen as 'consumers' of their input tasks, and 'producers' of their output tasks. In this perspective, the completion of an activity during a case can be seen as

[2] In First-order logic, the p/n notation is used to denote an n-ary predicate symbol p.

[3] Note that this is a different meaning than in Petri Nets.

the production of a resource[4] that is to be consumed by some transition. So, a first kind of limitation to the possible combinations of transitions is expressed by WoMan using the following predicate:

– `transition_provider` $([\tau_1, \ldots, \tau_n], t, q)$: transition t, involving input tasks $I = [i_1, \ldots, i_n]$, is enabled provided that each task $i_k \in I, k = 1, \ldots, n$ was 'produced' as an output of transition τ_k, where the τ_k's are placeholders (variables) to be interpreted according to the Object Identity assumption ("terms (even variables) denoted with different symbols must be distinct (i.e., they must refer to different objects)"); several combinations of transition providers can be allowed, numbered by progressive q.

It partitions the input multiset of a transition according to the producers of the activities to be consumed. Let us see this through an example.

Example 1 Consider a model that includes, among others, the following transitions:

$$t_1 : \{x, y, z\} \Rightarrow \{a, b\} \quad ; \quad t_2 : \{x, y\} \Rightarrow \{a\} \quad ; \quad t_3 : \{x\} \Rightarrow \{a, d\}$$

and suppose that the current set of activities to be 'consumed' is $\{x, y, z\}$. If an activity a is started, any of the above transitions might be the 'consumer'. Suppose that WoMan also knows the producers of these activities: $\{x/p22, y/p21, z/p22\}$, and that the model includes the following atoms related to transitions t_1, t_2 and t_3:

`transition_provider` $([\tau_1, \tau_1, \tau_2], t_1, 1)$.
`transition_provider` $([\tau_1, \tau_2, \tau_2], t_1, 2)$.
`transition_provider` $([\tau_1, \tau_2, \tau_1], t_1, 3)$.
`transition_provider` $([\tau_1, \tau_1], t_2, 1)$.
`transition_provider` $([\tau_1], t_3, 1)$.

In this case, transition t_2 is not a valid consumer, since it would require that both x and y were produced by the same transition τ_1, while they were actually produced by two different transitions ($p22$ and $p21$, respectively). Conversely, pattern #3 of transition t_1 is compliant with the available producers, which makes it an eligible candidate. Also transition t_3 is enabled.

Additional constraints concern the agents that may run the activities:

– `task_agent` (t, A) : an agent, matching the roles A, can carry out task t.
– `transition_agent` $([A'_1, \ldots, A'_n], [A''_1, \ldots, A''_m], t, C_{tq}, q)$: transition t, involving input tasks $I = [i_1, \ldots, i_n]$ and output tasks $O = [o_1, \ldots, o_m]$, may occur provided that each task $i_k \in I, k = 1, \ldots, n$ is carried out by an agent matching roles A'_k, and that each task $o_j \in O$, $j = 1, \ldots, m$ is carried out by an agent matching roles A''_j; several combinations of transition agents can be allowed, numbered by progressive q, each encountered in cases C_{tq}.

WoMan can handle taxonomies of agent roles. Each A'_k or A''_j is an expression in disjunctive normal form:

$$(r_{11} \wedge \cdots \wedge r_{1n_1}) \vee \cdots \vee (r_{m1} \wedge \cdots \wedge r_{mn_m})$$

where each r_{ij} is an individual or a role in the taxonomy, meaning that the agent must match all roles in at least one disjunct. The conjuncts are introduced to handle multiple inheritance. The generalization/specialization relationship is handled, in that a role is considered

[4]In this perspective the term *resource* assumes a general meaning, which is not relative to the event resource R that executes an activity in a process case.

as matched by an agent if the agent matches any of its subclasses in the taxonomy. During the mining phase, generalizing means replacing one or more roles/instances with one of their superclasses.

More recent versions of the WoMan formalism (Ferilli et al. 2017a, b) added the following predicates to deal with time constraints:

- `task_time` $(t, [b', b''], [e', e''])$: task t must begin at a time $i_b \in [b', b'']$ and end at a time $i_e \in [e', e'']$;
- `transition_time` $(t, [b', b''], [e', e''])$: transition t must begin at a time $i_b \in [b', b'']$ and end at a time $i_e \in [e', e'']$;
- `task_in_transition_time` $(t, p, [b', b''], [e', e''])$: task t, when run in the output set of transition p, must begin at a time $i_b \in [b', b'']$ and end at a time $i_e \in [e', e'']$;

where i_b, b', b'', i_e, e', and e'' are relative to the start of the process execution, i.e. they are computed as the timestamp difference between the **begin_process** event and the event they refer to.

In addition to the exact timestamp of events, WoMan internally associates each activity in a case to a unique integer identifier, called *step*, assigned by progressive start timestamp. So, the above constraints may be expressed also in terms of steps, as follows:

- `task_step` $(t, [b', b''], [e', e''])$: task t must start at a step $s_b \in [b', b'']$ and end at a step $s_e \in [e', e'']$;
- `transition_step` $(t, [b', b''], [e', e''])$: transition t must start at a step $s_b \in [b', b'']$ and end at a step $s_e \in [e', e'']$;
- `task_in_transition_step` $(t, p, [b', b''], [e', e''])$: task t, when run in the output set of transition p, must start at a step $s_b \in [b', b'']$ and end at a step $s_e \in [e', e'']$;

These temporal constraints are mined on the entire training set. Begin and end times are relative to the start of the process execution, computed as the timestamp difference between the begin of process and the event they refer to. Step information is computed on the progressive number s on which a task t is executed.

Example 2 Consider, for instance, task $act\_Meal\_Preparation$ and transition

$$p23 : \{act\_Meal\_Preparation\} \Rightarrow \{act\_Meal\_Preparation, act\_Relax\}$$

An example of the new components for them might be:

```
task_time(act_Meal_Preparation,[5,10],[10,21])
transition_time(p23,[6,8],[20,25])
task_step(act_Meal_Preparation,[s3,s5],[s7,s12])
transition_step(p23,[s4,s5],[s10,s13])
task_in_transition_step(act_Meal_Preparation,p23,[s10,s11],[s11,s12])
task_in_transition_time(act_Meal_Preparation,p23,[10,12],[18,21])
```

Finally, WoMan can express pre- and post-conditions for tasks (in general), transitions, and tasks in the context of a specific transition. Specifically, conditions on transitions define when a transition may take place; task conditions define what must be true for a given task in general, task in transition conditions define further constraints for allowing a task to be run in the context of a specific transition (provided that its general conditions are met). They are defined as FOL rules of the following form:

- `act_`T (A, S, R) `:-` ... meaning that "activity A, corresponding to task T, can be run by agent R at step S of a case execution provided that ...";

– `trans_`T`(`S`) :- ...` meaning that "transition T can be run at step S of a case execution provided that ..."";
– `act_`T`_in_trans_`P`(`A`,` S`,` R`) :- ...` meaning that "activity A, of type T, can be run by agent R in the context of transition P at step S of a case execution provided that ...";

where the premises '...' are conjunctions of atoms based on contextual and control flow information. Conditions are not limited to the current status of execution. They may involve the status at several steps using two predicates:

– `activity(`s`,` t`)` : at step s (unique identifier), $t \in \mathcal{T}$ is executed;
– `after(`s'`,` s''`, [`n'`,` n''`] , [`m'`,` m''`])` : step s'' follows step s' after a number of steps ranging between n' and n'' and after a time ranging between m' and m''.

Due to concurrency, predicate `after/3` induces a partial ordering on the set of steps. The difference between pre- and post- conditions is that premises in the former refer only to steps up to S, while in the latter they may refer to any step, both before and after S.

4 Heuristic activity prediction in WoMan

In order to describe WoMan's activity prediction functionality, we must first introduce WoMan's supervision module, called **WEST**. Indeed, predictions are based on the current status of the process enactment as computed by this module.

4.1 Supervision procedure

WEST (Workflow Enactment Supervisor and Trainer) (Ferilli et al. 2016) checks whether new cases are compliant with a given process model $\mathcal{M}$. More specifically, given the current status S and a new event e, the compliance check of the latter to the former may yield 3 possible outcomes:

ok : e is compliant with S;
error : e causes a syntactic inconsistency (e.g., it terminates an activity that was never started, or completes a case while activities are still running); or
warning : indicating a deviation from the model that does not violate syntactic constraints; more specifically, the following types of warnings are available:

1. the pre-/post-conditions of a task, a transition or a task in the context of a transition are not fulfilled;
2. unexpected agent running a certain activity, in general or in the context of a specific transition;
3. a known task or transition, not expected at the current point of process execution, was run;
4. a new task or transition was run;
5. a task or transition was run more times than expected;
6. a task, transition or task in the context of a specific transition started or ended out of the expected time or step bounds.

Each warning carries a different degree of severity, expressed as a numeric weight. E.g., an unexpected task or transition also implies that the agent that runs it was not expected,

and so has a greater severity degree than the unexpected agent alone. The degrees of severity currently embedded in WoMan for each type of warning were heuristically determined (a discussion and experimentation on how to determine these weights in order to improve performance is outside the scope of this paper).

Algorithm 1 Maintenance of the structure recording valid statuses in WEST

Require: $\mathcal{M}$: process model
Require: $\mathcal{S}$: set of currently compliant statuses compatible with the case
Require: *Running* : set of currently running activities
Require: *Transitions*: list of transitions actually carried out so far
Require: $\langle T, E, W, P, A, O, R\rangle$: log entry
1: *Activities* $\leftarrow \emptyset$ /* Multiset of started activities */
2: **if** E = begin activity **then**
3: *Activities* $\leftarrow$ *Activities* $\cup \{A\}$
4: *Running* $\leftarrow$ *Running* $\cup \{A\}$
5: **for all** $S = \langle M, R, C, T, P\rangle \in \mathcal{S}$ **do**
6: $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S\}$
7: **if** $A \in R$ **then**
8: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\langle M, R \setminus \{A\}, C, T, P\rangle\}$
9: **for all** $p : I \Rightarrow O\ [C_p] \in \mathcal{M} : A \in O$ **do**
10: **if** $\exists$`transition provider`$(Q, p, q) \in \mathcal{M}$: matches(Q, M) **then**
11: $P' \leftarrow P \cup P_{A,p,S}$ /* $P_{A,p,S}$ warnings raised by running A in p given S */
12: $R' = \{t'/p \mid t' \in O \wedge t' \neq A\} \cup R$
13: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\langle M \setminus I, R', C \cap C_p, T\&\langle p\rangle, P'\rangle\}$
14: **if** E = end activity **then**
15: **if** $A \notin$ *Running* **then**
16: Error
17: **else**
18: *Running* $\leftarrow$ *Running* $\setminus \{A\}$
19: **for all** $S = \langle M, R, C, T, P\rangle \in \mathcal{S}$ **do**
20: select transition $t : I \Rightarrow O \in T$ that produced A
21: $S \leftarrow \langle M \cup \{A/t\}, R, C, T, P\rangle$
22: **if** a transition t has been fully carried out **then**
23: Transitions $\leftarrow$ Transitions $\&\langle t\rangle$
24: **for all** $S = \langle M, R, C, T, P\rangle \in \mathcal{S}$ **do**
25: **if** $T \neq$ Transitions **then**
26: $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S\}$

where matches(Q, M) checks that provider constraint Q is fulfilled by marking M.

It is easy to note that different transitions in a model may be composed in different ways with each other. So, a partial process execution might be compliant to several different exploitations of the model. As a consequence, many transitions may be eligible for application at any moment, and when a new activity takes place there may be some ambiguity about which one is actually being fired. This is clear from Example 1, where each of the two proposed options would change in a different way the status of the process, as follows: firing t_1 would consume x, y and z, leaving no activity to be consumed and causing the system to wait for a later activation of b, 'produced' by t_1; firing t_2 is inhibited, because the

transition providers do not match the required pattern of variables (if it were enabled, firing it would consume x and y, leaving $\{z/p22\}$ to be consumed and causing the completion of transition t_2); firing t_3 would consume x, leaving $\{y/p21, z/p22\}$ to be consumed and causing WoMan to wait for a later activation of d, 'produced' by t_3.

We call each of these alternatives a *status*. This ambiguity about different statuses that are compliant with a model at a given time of process enactment must be properly handled when supervising the process enactment. Since it can be resolved only at a later time, all the corresponding alternate evolutions of the status must be carried on by the system, and each new event must be handled with respect to each alternate status. Then, in some cases, the same ambiguity issues will arise, and more alternate evolutions will be generated; in other cases, the new event will point out that some current alternate statuses were wrong, and will cause them to be dropped. So, as long as the process enactment proceeds, the set of alternate statuses that are compliant with the activities carried out so far can be both expanded with new branches, and pruned of all alternatives that become incompatible with the activities carried out so far.

To handle this ambiguity, WEST maintains the set $\mathcal{S}$ of statuses, each represented as a 5-tuple of sets $\langle M, R, C, T, W\rangle$ recording the following information:

- M the marking, i.e., terminated activities, not yet used to fire a transition;
- R (for 'Ready') the output activities of fired transitions in the status, and that the system is waiting for in order to complete them;
- C training cases that are compliant with that status;
- T (hypothesized) transitions that have been fired to reach that status;
- W multiset of warnings raised by the various events that led to that status.

As long as the process executions proceeds, new alternative statuses may be added to $\mathcal{S}$, and statuses that are not compliant with the model may be removed. The way in which WoMan maintains $\mathcal{S}$ is specified in Algorithm 1, which requires a process model $\mathcal{M}$ to be exploited, the list of actually fired transitions *Transitions*, the set $\mathcal{S}$ of currently valid statuses, the set of currently running tasks *Running*, and the entry $\langle T, E, W, P, A, O, R\rangle$ of the new case event to be checked. The algorithm works in two settings, depending on the event type E. If E concerns the begin of activity A, then it is stored in the multiset of started activities *Activities*,[5] and each status $S = \langle M, R, C, T, P\rangle \in \mathcal{S}$ is removed and evaluated. The evaluation of a status S occurs in two ways and it can lead to the generation of more alternate statuses. The first way is to generate an evolution from the removal of an occurrence of activity A from the R component of S, if it was already waiting A as an exptected output of a previous applied transition. The second way is to generate an evolution for each marked transition $p : I \Rightarrow O$ of the model $\mathcal{M}$, that involves activity A in its output set O. A transition p is marked if there exists at least one transition provider q whose set Q (containing input tasks with producer placeholders) matches, under the Object Identity assumption, the M component of S. Status S is pruned if there is no way to evolve it. If E concerns the end of activity A, then the set of the running activities *Running* is updated by removing A and for each status S in $\mathcal{S}$ the M component is updated by adding A and its provider t, since it is a new token. If t is completed, the list of transitions *Transitions* is updated by adding t. In order to avoid uncompliant statuses, WEST checks for each status S in the updated $\mathcal{S}$ whether the list of transitions T of S is compliant with the list *Transitions*, otherwise it prunes the status S from $\mathcal{S}$.

[5] It is a multiset since an activity could occur more than once.

4.2 Statistics maintenance

The activity prediction module of WoMan, **SNAP** (Suggester of Next Activity in Process), exploits $\mathcal{S}$ (maintained by WEST) to compute statistics that are useful to determine which are the expected next activities and to rank them by some sort of likelihood according to Algorithm 2. For each surviving status $S \in \mathcal{S}$ at a given moment of process execution, its *discrepancy* from the model, $\delta(S)$, can be computed based on W, to be used in the prediction procedure. Since each status may be associated with different activities to be performed next, there is also an ambiguity about which activities will be carried out. In a preliminary phase each $S \in \mathcal{S}$ is removed and evaluated. For each transition in the model enabled by the M (Marking) component of S, according to the transition provider constraint, the evolution S' of S is added to $\mathcal{S}$, and W is updated if inconsistency from observed and learned behavior is encountered. Then, the discrepancy $\delta(S)$ of each $S \in \mathcal{S}$ is measured, and statuses that exceed a certain discrepancy tolerance threshold ϵ are removed. Based on the statuses with a low discrepancy only, the set $\mathcal{N}$ of activities that may be carried out next, with their transition provider, is selected from the R (Ready) component of each $S \in \mathcal{S}$. The aim is ranking the tasks in $\mathcal{N}$ by decreasing likelihood, so that the resulting ranking can be used as a prediction.

Algorithm 2 Activity Prediction in WoMan using SNAP

Require: $\mathcal{M}$: process model
Require: $\mathcal{S}$: set of currently compliant statuses compatible with the case
Require: E : current event of trace
Require: ϵ: threshold to filter only more compliant statuses

1: **if** E = end_activity $\vee$ E = begin_process **then**
2: $\quad\mathcal{S}' \leftarrow \emptyset$
3: $\quad$**for all** $S = \langle M, R, C, T, P\rangle \in \mathcal{S}$ **do**
4: $\quad\quad$**for all** $p : I \Rightarrow O\ [C_p] \in \mathcal{M}$ **do**
5: $\quad\quad\quad$**if** $\exists$`transition_provider`$(Q, p, q) \in \mathcal{M}$: matches(Q, M) **then**
6: $\quad\quad\quad\quad P' \leftarrow P \cup P_{p,S}$ /* $P_{p,S}$ warnings raised by firing p given S */
7: $\quad\quad\quad\quad \mathcal{S}' \leftarrow \mathcal{S}' \cup \{\langle M \setminus I, R \cup O, C \cap C_p, T\&\langle t\rangle, P'\rangle\}$
8: **if** E = begin_activity **then**
9: $\quad\mathcal{S}' \leftarrow \mathcal{S}$
/* multiset of candidate next activities */
10: $\mathcal{N} = \{\langle a, p'\rangle \mid \exists S = \langle M, R, C, T, P\rangle \in \mathcal{S}' : \delta(S) < \epsilon \wedge \langle a, p'\rangle \in R\}$
11: $\mathcal{R} \leftarrow$ *rankingAlgorithm*$(\mathcal{S}', \mathcal{N})$
12: $\mathcal{R}' \leftarrow \{\langle$ score,b,a $\rangle \mid \langle$ score,a $\rangle \in \mathcal{R}, \langle a, p'\rangle \in \mathcal{N} \wedge b = b' + b''$ such that $\exists$ task_in_transition_step$(a, p',[b', b''],[e', e''])\ \}$

where matches(Q, M) checks that provider constraint Q is fulfilled by marking M.

Finally, in case of ties in the ranking, i.e., activities with equal score, they are further discriminated using statistics about their starting step. Since a task t in the ranking is expected to be executed in the context of a certain transition p, a step score is computed on its $task\_in\_transition\_step(t, p, [b', b''], [e', e''])$. The underlying idea is that the lower the sum b of the minimum and maximum begin steps b' and b", the higher the chances of being executed before the others (this is especially true for parallel tasks). In the following we describe in detail two approaches to obtain the final ranking of predictions, the Heuristic one in Section 4.3 and the Bayesian one in Section 5.

4.3 Heuristic approach

The traditional way in which WoMan produces activity predictions works as specified in Algorithm 3. Specifically, activities in $\mathcal{N}$ are scored and ranked based on a heuristic combination of the following parameters, computed over $\mathcal{S}$:

1. $\mu_a(\mathcal{S})$, multiplicity of a across the various statuses $\mathcal{S}$ (activities that appear in more statuses are more likely to be carried out next);
2. C_a, number of cases with which the status that includes a is compliant (activities expected in the statuses supported by more training cases are more likely to be carried out next);
3. $\delta_a(S)$, sum of weights of warnings raised by the status in which the activity is included (activities expected in statuses that raised less warnings are more likely to be carried out next).

Algorithm 3 Heuristic Ranking

Require: $\mathcal{S}$: set of compliant statuses evolved by SNAP
Require: $\mathcal{N}$: set of ready to be enabled activities
1: $\mathcal{R} \leftarrow \emptyset$
2: **for all** $\langle a, p' \rangle \in \mathcal{N}$ **do**
3: $\quad \mathcal{S}_a = \{\langle M, R, C, T, W \rangle \in \mathcal{S} \mid \langle a, p' \rangle \in R\}$
4: $\quad \delta_a = \sum_{S \in \mathcal{S}_a} \delta(S)$ /* overall discrepancy of all statuses involving a */
5: $\quad C_a = \bigcup_{\langle M,R,C,T,P \rangle \in \mathcal{S}_a} C$ /* cases set supporting the a's execution in all statuses */
6: $\quad score \leftarrow (|C_a| \cdot |\mathcal{S}_a| \cdot |a|_{\mathcal{N}}) \,/\, \delta_a$
7: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{\langle score, a \rangle\}$
Ensure: $\mathcal{R}$

where:

- $|\cdot|$ denotes the cardinality of a set or multiset;
- $|\cdot|_M$ denotes the number of occurrences of an element in a multiset M;
- $\delta(\cdot)$ is the discrepancy of a status, computed as the sum of the weights of the warnings raised by the status.

5 Bayesian approach

The Bayesian approach we propose is inspired by the classical Naïve Bayes algorithm used in Machine Learning (Mitchell 1997), according to which

$$P(\theta|\mathbf{F}) = P(\theta)\frac{P(\mathbf{F}|\theta)}{P(\mathbf{F})} = P(\theta)\frac{\prod_{f \in \mathbf{F}} P(f|\theta)}{\prod_{f \in \mathbf{F}} P(f)} \tag{1}$$

(assuming independence between items in $\mathbf{F}$). It consists of two phases: the learning of the Bayesian model and a *Maximum A Posteriori* (MAP) estimation of the most plausible activity to be carried out next. In our case, the aim is computing the likelihood that an activity θ will be carried out next at a given moment of a process execution, given the values of the features $\mathbf{F}$ that describe the current status of process execution. So, we must define both the possible θ's and the set of features $\mathbf{F}$ that affect the posterior probability of θ.

Let us first define the set $\mathcal{T} \times \mathbb{N}$ of all possible pairs $\langle t, o \rangle$ where t is a task in the process model, and o is the progressive number of occurrence of that task in an execution.[6] We consider as θ's the set Θ of all the $\langle t, o \rangle \in \mathcal{T} \times \mathbb{N}$ that make sense in the current status of the process execution, i.e., such that $t \in \mathcal{N}$ (activity t is a candidate next activity) and running activity t at this point would be its o-th occurrence in the current process execution.

Concerning the features, we focus on control-flow based ones. Let us define the set $\Gamma \subset (\mathcal{T} \cup \mathcal{P}) \times \mathbb{N}$ of all possible pairs $\langle t, o \rangle$ where t can be either a task or transition in the model, and o is the progressive number of occurrence of t during the control-flow supervision (as shown in Algorithm 1). For each $\gamma \in \Gamma$, we consider a set $R = \{p, m, c\}$ of 3 types of relationships between a θ and a γ in a process case:

p (Parallel γ) occurrences of running tasks/transitions when θ was started:

if γ is a task, it was not terminated yet;
if γ is a transition, it is not completed yet;

m (Marked γ) occurrences of tasks/transitions that were terminated but not yet consumed before θ was started:

if γ is a task, it was not yet consumed by any transition;
if γ is a transition, it was fired but still has marked output tasks;

c (Consumed γ) occurrences of tasks/transitions that are terminated and have already been used, before θ was started:

if γ is a task, it has been used by some fired transition;
if γ is a transition, all its output tasks have already been used.

So, we define the set of features as $\mathbf{F} = R \times \Gamma$, and the value of each $f \in \mathbf{F}$ as its frequency in the training cases. Let us see this through an example.

Example 3 Consider the process case $c1$ in Fig. 1, where nodes represent *steps* (unique identifiers associated to activities based on their progressive order of execution), labeled with pairs $\langle t, o \rangle$ (indicating that they are associated to the the o-th occurrence of task t), and edges represent the *next* relation between pairs of steps. Suppose that the model includes, among others, the following transitions:

$t0 : \{start\} \Rightarrow \{a\}$

$t1 : \{a\} \Rightarrow \{b, c, d\}$

$t2 : \{b\} \Rightarrow \{e\}$

$t3 : \{e\} \Rightarrow \{b, c\}$

$t4 : \{b, c, c\} \Rightarrow \{f\}$

$t5 : \{d, f\} \Rightarrow \{stop\}$

For each step, information about relationships is maintained. Suppose $c1$ was partially executed up to $s8$ (not included, i.e., before $\langle f, 1 \rangle$ is started); then, $\mathbf{F}$ would contain:

(p) Parallel γ's :

[6] This is a relevant difference with respect to the heuristic approach, where there is no distinction about different occurrences of the same task. Our expectation is that, by considering such a more fine-grained information, predictions may be more accurate.

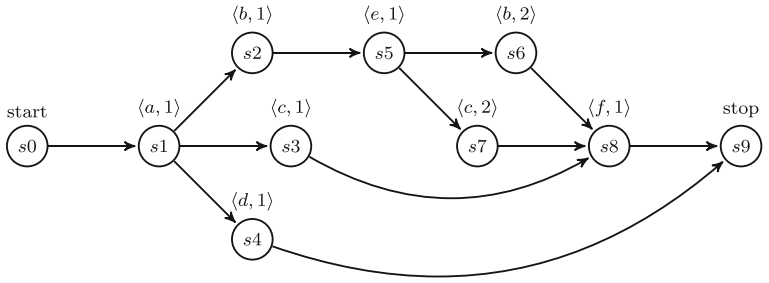

Fig. 1 c_1: Example of process case c1

for tasks = $\{\langle d, 1\rangle\}$
for transitions = $\{\langle t1, 1\rangle\}$

(c) Consumed γ's :

for tasks = $\{\langle a, 1\rangle\ \langle b, 1\rangle\ \langle e, 1\rangle\}$
for transitions = $\{\langle t0, 1\rangle\ \langle t2, 1\rangle\}$

(m) Marked γ's :

for tasks = $\{\langle c, 1\rangle\ \langle b, 2\rangle\ \langle c, 2\rangle\}$
for transitions = $\{\langle t1, 1\rangle\ \langle t3, 1\rangle\}$

In this case, the first occurrence of transition $t1$ is both parallel and marked, since $\langle d, 1\rangle$ is a *running* task and $\langle c, 1\rangle$ is a token to enable something (in this case $\langle f, 1\rangle$). The first occurrence of transition $t3$ ($\langle t3, 1\rangle$) is marked, since $\langle b, 2\rangle$ and $\langle c, 2\rangle$ should be consumed by the activation of $\langle f, 1\rangle$. Transitions $\langle t0, 1\rangle$ and $\langle t2, 1\rangle$ are terminated and consumed since their output tasks ($\{\langle a, 1\rangle\}$ and $\{\langle e, 1\rangle\}$) are all consumed. Task $\{\langle b, 1\rangle\}$ is consumed, because it was used to fire transition $\{\langle t2, 1\rangle\}$. And so on.

After the case terminates, features **F** are used to train the Bayesian model by updating their frequencies. In the classification step, they will be used by extracting from the model the corresponding frequency value.

Thus, learning the control-flow based Bayesian model corresponds to compute a 3-dimensional tensor $\mathcal{B} \in \mathbb{R}^{|R|\times|\Theta|\times|\Gamma|}$, where $|\cdot|$ denotes the cardinality of a set. A point $(r, \theta, \gamma) \in \mathcal{B}$ is the frequency with which relationship r occurs between θ and γ over the historical cases. Since the occurrence (firing/termination) of transitions is known during the process discovery phase, we compute/update the frequencies (i.e., the feature values) as long as the discovery algorithm, applied to a case, proceeds as described in (Ferilli 2014). For each step s in the process case, the associated pair $\langle t, o\rangle$ is considered as a θ, each pair $\langle t, o\rangle$ associated to a previous step is considered as a γ, and the tensor is updated by incrementing all points (r, θ, γ) such that relationship r holds between θ and γ.[7]

So, the components of (1), at a certain partial status of the process execution, can be computed as follows:

$P(\theta)$ is the prior probability of θ; it is trivially defined as the fraction of all cases in which θ was observed.

[7] In each case, a relationship between a γ and a θ occurs at most once, if any.

$P(f|\theta)$ is the likelihood that feature $f = (r, \gamma)$ is observed if θ were executed; it is obtained as the value at coordinates (r, θ, γ) in tensor $\mathcal{B}$ with the triple (r, θ, γ).

$P(f)$ is the evidence of feature f, regardless of θ; this probability is obtained by querying the matrix $\mathcal{A} \in \mathbb{R}^{|R|\times|\Gamma|}$, obtained by applying the aggregation operator to remove dimension Θ from $\mathcal{B}$, with the pair (r, γ).

In our case, many values would be 0, which would make 0 also the product on the right-hand-side of (1), and thus also its left-hand-side. To avoid this, we consider in our computation only non-0 probabilities, which leads to the following final form of (1):

$$P(\theta|\mathbf{F}) = P(\theta)\frac{\prod_{f\in\mathbf{F}} P(f|\theta)}{\prod_{f\in\mathbf{F}} P(f)} = P(\theta)\frac{\prod_{(r,\gamma)\in\mathbf{F},\mathcal{B}_{r,\theta,\gamma}\neq 0} \mathcal{B}_{r,\theta,\gamma}}{\prod_{(r,\gamma)\in\mathbf{F},\mathcal{A}_{r,\gamma}} \mathcal{A}_{r,\gamma}} \tag{2}$$

The learned Bayesian model can be exploited during the supervision task for the *operational support*, i.e., the prediction of the next activity. Given the current status S of a partial process execution, and the set Θ_S of next plausible activities associated to S, the next most plausible activity $\overline{\theta}$ is chosen by applying the MAP strategy:

$$\overline{\theta} = \arg\max_{\theta\in\Theta_S} P(\theta|\mathbf{F}) \tag{3}$$

This approach can be easily embedded in **SNAP**, using the complementary of the normalized computed posterior probability as the score to rank the candidate ready activities in decreasing order.

6 Evaluation

The performance of the proposed activity prediction approaches was evaluated on several datasets, concerning different kinds of processes associated with different kinds and levels of complexity. The datasets related to Ambient Intelligence concern typical user behavior. Thus, they involve much more variability and subjectivity than in industrial processes, and there is no 'correct' underlying model, just some kind of 'typicality' can be expected:

Aruba from the CASAS benchmark repository.[8] It includes continuous recordings of home activities of an elderly person, visited from time to time by her children, in a time span of 220 days. Each day is mapped to a case of the process representing the daily routine of the elderly person. Transitions correspond to terminating some activities and starting new activities. The resources (persons) that perform activities are unknown.

GPItaly from one of the Italian use cases of the GiraffPlus project[9] (Coradeschi et al. 2013). It concerns the movements of an elderly person (and occasionally other people) in the various rooms of her home along 253 days. Each day is a case of the process representing the typical movements of people in the home. Tasks correspond to rooms; transitions correspond to leaving a room and entering another.

The other concerns chess playing as a General Game Playing, where again the 'correct' model is not available:

[8] http://ailab.wsu.edu/casas/datasets.html

[9] http://www.giraffplus.eu

Table 1 Dataset statistics

	Cases	Events		Activities		Tasks		Transitions	
		overall	avg	overall	avg	overall	avg	overall	avg
Aruba	220	13788	62.67	6674	30.34	10	0.05	92	0.42
GPItaly	253	185844	369.47	92669	366.28	8	0.03	79	0.31
White	994	239104	240.55	118566	119.28	724	0.73	11694	11.76
Black	527	129794	246.29	64370	122.14	725	1.37	8934	16.95
Draw	1408	302634	214.94	149909	106.47	731	0.52	11938	8.48

Chess from the Italian Chess Federation website.[10] 2929 reports of actual top-level matches were downloaded. Each match is a case, belonging to one of 3 processes associated to the possible match outcomes: *white* wins, *black* wins, or *draw*. A task is the occupation of a square by a specific kind of piece (e.g., "black rook in a8"). Transitions correspond to moves: each move of a player terminates some activities (since it moves pieces away from the squares they currently occupy) and starts new activities (that is, the occupation by pieces of their destination squares). The involved resources are the two players: 'white' and 'black'.

Table 1 reports statistics on the experimental datasets: number of cases and number of events, activities, tasks and transitions, also on average per case. There are more cases for the chess datasets than for the Ambient Intelligence ones. However, the chess datasets involve many more different tasks and transitions, many of which are rare or even unique. The datasets are different also from a qualitative viewpoint. Aruba cases feature many short loops and some concurrency (involving up to 2 activities), optional and duplicated activities. The same holds for GPItaly, except for concurrency. The chess datasets are characterized by very high concurrency: each game starts with 32 concurrent activities, a number which is beyond the reach of many current process mining systems (Ferilli and Esposito 2013). This number progressively decreases (but remains still high) as long as the game proceeds. Short and nested loops, optional and duplicated tasks are present as well. The number of agents and temporal constraints is not shown, since the former is at least equal, and the latter is exactly equal, to the number of tasks and transitions.

The experimental procedure was as follows. First, each dataset was translated from its original representation to the input format of WoMan. Then, a 10-fold cross-validation procedure was run for each dataset, using the learning functionality of WoMan proposed in Ferilli (2014) to learn models for all training sets. Finally, each model was used as a reference to call WEST and SNAP on each event in the test sets: the former checked compliance of the new event and suitably updated the set of statuses associated to the current case, while the latter used the resulting set of statuses to make a prediction about the next activity that is expected in that case. Specifically, the following settings were tried:

H Heuristic ranking only;
B Bayesian ranking only;
H-B Heuristic ranking, followed by Bayesian ranking to resolve ties;
B-H Bayesian ranking, followed by Heuristic ranking to resolve ties.

[10] http://scacchi.qnet.it

Table 2 Activity prediction statistics (10-fold Cross Validation)

	P	R	Rank				Tasks
H vs B			H	B	H-B	B-H	
Aruba	0.89	0.94	0.72	0.72	**0.77**	**0.77**	5.4
GPItaly	1.0	0.98	0.53	**0.66**	0.63	**0.66**	7.57
black	0.62	0.90	**0.97**	0.96	0.96	0.96	10.18
white	0.69	0.90	**0.95**	0.94	**0.95**	**0.95**	9.8
draw	0.79	0.93	**0.96**	0.95	**0.96**	**0.96**	9.55
chess	0.70	0.91	**0.96**	0.95	**0.96**	**0.96**	9.84

Table 2 reports the processes on the row headings ('chess' referring to the average of the chess sub-datasets), and corresponding average performance for several settings and measures on the columns. Column P (for *Predictions*) reports the ratio of cases in which SNAP returned a prediction. Indeed, when tasks or transitions not present in the model are executed in the current enactment, WoMan assumes a new kind of process is enacted, and avoids making predictions. Column R (for *Recall*) reports the ratio of cases in which the correct activity (i.e., the activity that is actually carried out next) is present in the ranking, among those in which a prediction was made. Finally, column *Rank* reports how close it is to the first element of the ranking (1.0 meaning it is the first in the ranking, and 0.0 meaning it is the last in the ranking), and *Tasks* is the average length of the ranking (the lower, the better). Note that values of P, R and *Tasks* are independent from the involved approaches (**H** or **B**). To provide an immediate indication of the overall activity prediction performance, Table 3 reports the values of a global index, called *Quality* and defined as:

$$Quality = Pred \cdot Recall \cdot Rank \in [0, 1]$$

$Quality = 0$ means that predictions are completely unreliable; $Quality = 1$ means that WoMan always makes a prediction, and that such a prediction is correct (i.e., the correct activity is at the top of the ranking). In Tables 2 and 3 the winning settings are reported in bold.

First, note that *Quality* index in the hybrid approaches (**H-B** or **B-H**) is not worse than in the single approaches in 5 out 6 cases. This is due to the combination's ability to better discriminate the suggested tasks in the ranking. As regards the *Rank* measure, the Heuristic approach outperforms the Bayesian one in all datasets except for GPItaly. Since the GPItaly dataset involves much more events and activities than the Aruba one, on a comparable number of cases, tasks, and transitions, it is more profitable for the Naïve Bayes model to collect more meaningful statistics. However, the heuristic approach works very well even if the dataset has a lower distribution. As regards performance on the chess dataset, there's no difference among all approaches. This is probably due to the large volume of available data.

WoMan is extremely reliable because the correct next activity (measured by R) is almost always present in the ranking (90-98% of the times), and often in the top section of it: more specifically, it is in the top 10% items for the chess processes, and in the top 30% items for the Aruba dataset. This confirms that WoMan is effective under very different conditions as regards the complexity of the models to be handled. In the Ambient Intelligence domain,

Table 3 Activity prediction quality (10-fold Cross Validation)

H vs B	H	B	H-B	B-H
Aruba	0.60	0.60	**0.62**	**0.62**
GPItaly	0.51	0.64	**0.65**	0.4
black	**0.54**	0.53	0.53	0.53
white	**0.59**	**0.59**	**0.59**	**0.59**
draw	**0.71**	0.70	**0.71**	**0.71**
chess	**0.61**	**0.61**	**0.61**	**0.61**

this means that it may be worth spending some effort to prepare the environment in order to facilitate that activity, or to provide the user with suitable support for that activity. In the chess domain, this provides a first tool to make the machine able to play autonomously. The number of predictions is proportional to the number of tasks and transitions in the model. This was expected, because, the more variability in behaviors, the more likely it is that the test sets contain behaviors that were not present in the training sets. Compared to previous results, the number of predictions, in all domains, increased, due to the fact that the 10-fold cross-validation procedure involves larger training sets, providing more information to the learning step. WoMan is almost always able to make a prediction in the Ambient Intelligence domain, which is extremely important in order to provide continuous support to the users. While neatly lower, the percentage of predictions in the chess domain was clearly improved, and covers more than half of the match. Interestingly, albeit the evaluation metrics are different and not directly comparable, the *Quality* is neatly above the state-of-the-art performance obtained using Deep Learning (Lai 2015) and Neural Networks (Oshri and Khandwala 2016). The nice thing is that WoMan reaches this percentage by being able to distinguish cases in which it can make an extremely reliable prediction from cases in which it prefers not to make a prediction at all.

7 Conclusions

In addition to other classical exploitations, process models may be used to predict the next activities that will take place. This would allow to take suitable actions to help accomplishing those activities, which is particularly important when the processes involve very variable and flexible behavior. This paper proposed and compared two approaches to make these kinds of predictions using the WoMan framework for workflow management. Experimental results on different tasks and domains suggest that the proposed approaches can successfully perform such predictions, and highlighting the pros and cons of each and the advantages of using them in combination. This further confirms that WoMan is suitable to handle several process-related tasks in complex domains.

Future work will concern the application of the proposed framework and approaches to other domains, e.g. Industry 4.0 ones, and the embedding of the prediction module in other applications, in order to guide their behavior.

References

Agrawal, R., Gunopulos, D., Leymann, F. (1998). Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*. https://doi.org/10.1007/BFb0101003.

Ceci, M., Lanotte, P.F., Fumarola, F., Cavallo, D.P., Malerba, D. (2014). Completion time and next activity prediction of processes using sequential pattern mining. In *International Conference on Discovery Science* (pp. 49–61): Springer. https://doi.org/10.1007/978-3-319-11812-3_5.

Ceri, S., Gottlob, G., Tanca, L. (1990). *Logic Programming and Databases*. Verlag: Springer. https://doi.org/10.1007/978-3-642-83952-8.

Cook, J., & Wolf, A. (1996). Discovering models of software processes from event-based data. Tech. Rep CU-CS-819-96, Department of Computer Science, University of Colorado. https://doi.org/10.1145/287000.287001.

Coradeschi, S., Cesta, A., Cortellessa, G., Coraci, L., Gonzalez, J., Karlsson, L., Furfari, F., Loutfi, A., Orlandini, A., Palumbo, F., Pecora, F., von Rump, S., Štimec, U.J., Tslund, B. (2013). Giraffplus: Combining social interaction and long term monitoring for promoting independent living. In *Proceedings of the 6th international conference on human system interaction (HSI)* (pp. 578–585): IEEE. https://doi.org/10.1109/HSI.2013.6577883.

Ferilli, S., & Esposito, F. (2013). A logic framework for incremental learning of process models. *Fundamenta Informaticae*, *128*, 413–443. https://doi.org/10.3233/FI-2013-951.

Ferilli, S. (2014). Woman: Logic-based workflow learning and management. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *44*(6), 744–756. https://doi.org/10.1109/TSMC.2013.2273310.

Ferilli, S., Esposito, F., Redavid, D., Angelastro, S. (2016). Predicting process behavior in woman. In *AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016* (pp. 308–320): Proceedings. https://doi.org/10.1007/978-3-319-49130-1_23.

Ferilli, S., Esposito, F., Redavid, D., Angelastro, S. (2017a). Extended process models for activity prediction. In Kryszkiewicz, M., Appice, A., Slezak, D., Rybinski, H., Skowron, A., Rás Z (Eds.) *Foundations of Intelligent Systems, Springer, Lecture Notes in Artificial Intelligence*, (Vol. 10352 pp. 194–208). https://doi.org/10.1007/978-3-319-60438-1_36.

Ferilli, S., Redavid, D., Angelastro, S. (2017b). Activity prediction in process management using the woman framework. In Perner, P. (Ed.) *Advances in Data Mining. Applications and Theoretical Aspects, Springer, Lecture Notes in Artificial Intelligence*, (Vol. 10357 pp. 194–208). https://doi.org/10.1007/978-3-319-62701-4_15.

Greco, G., Guzzo, A., Pontieri, L. (2005). Mining hierarchies of models: From abstract views to concrete specifications. *Business Process Management*, *3649*, 32–47. https://doi.org/10.1007/11538394_3.

Herbst, J., & Karagiannis, D. (1998). Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. In *1998 Proceedings. Ninth International Workshop on Database and expert systems applications* (pp. 745–752): IEEE. https://doi.org/10.1109/DEXA.1998.707491.

Herbst, J., & Karagiannis, D. (1999). An inductive approach to the acquisition and adaptation of workflow models. In *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business* (pp 52–57), Stockholm, Sweden.

IEEE Task Force on Process Mining (2012). Process mining manifesto. In *Business Process Management Workshops, Lecture Notes in Business Information Processing* (vol. 99, pp. 169–194). https://doi.org/10.1007/978-3-642-28108-2_19.

Lai, M. (2015). Giraffe: Using deep reinforcement learning to play chess. CoRR 1509.01549.

Lloyd, J.W. (1987). Foundations of Logic Programming, 2nd edn., Springer, Berlin. https://doi.org/10.1007/978-3-642-96826-2.

Mitchell, T.M. (1997). *Machine Learning*, 1st edn. New York: McGraw-Hill, Inc. https://doi.org/10.1007/978-3-662-12405-5.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, *8*(4), 295–318. https://doi.org/10.1007/BF03037089.

Oshri, B., & Khandwala, N. (2016). Predicting moves in chess using convolutional neural networks. In *Stanford University Course Project Reports – CS231n: Convolutional Neural Networks for Visual Recognition*. https://cs231n.stanford.edu/reports/ConvChess.pdf.

Pesic, M., & van der Aalst, W.M.P. (2006). A declarative approach for flexible business processes management. In *Proceedings of the 2006 international conference on Business Process Management Workshops, BPM'06* (pp. 169–180): Springer-Verlag. https://doi.org/10.1007/11837862_18.

Schonenberg, H., Weber, B., van Dongen, B., van der Aalst, W. (2008). Supporting flexible processes through recommendations based on history. In: International Conference on Business Process Management (pp. 51–66): Springer. https://doi.org/10.1007/978-3-540-85758-7_7.

van der Aalst, W. (1998). The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, *8*, 21–66. https://doi.org/10.1142/S0218126698000043.

van der Aalst, W., Weijters, T., Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, *16*, 1128–1142. https://doi.org/10.1109/TKDE.2004.47.

van der Aalst, W.M., De Medeiros, A.A., Weijters, A. (2005). Genetic process mining. In *International Conference on Application and Theory of Petri Nets* (pp 48–69): Springer. https://doi.org/10.1007/11494744_5.

Weijters, A., & van der Aalst, W. (2001). Rediscovering workflow models from event-based data. In *Proceedings of the 11th dutch-belgian conference of machine learning (benelearn 2001)* (pp. 93–100).

Wen, L., Wang, J., Sun, J. (2006). Detecting implicit dependencies between tasks from event logs. *Frontiers of WWW Research and Development-APWeb 2006* (pp. 591–603). https://doi.org/10.1007/11610113_52.