Mining Chess Playing as a Complex Process

Stefano Ferilli[1], Domenico Redavid[2], and Sergio Angelastro[1]

[1] Dipartimento di Informatica, Università di Bari, Bari, Italy
[2] Artificial Brain S.r.l., Bari, Italy
stefano.ferilli@uniba.it

Abstract. The main objective of this paper is checking whether, and to what extent, advanced process mining techniques can support efficient and effective knowledge discovery in complex domains. For this purpose, we focused on chess playing, and cast it as a process. A secondary objective is checking whether the discovered information can provide interesting insight in the game rules and strategies, and/or may support effective game playing in future matches. Experimental results provide a positive answer to the former question, and encouraging clues on the latter.

1 Introduction

A *process* consists of a combination of actions, performed by any kind of agents, that modify the state of the process [2, 1]. It may involve sequential, parallel, conditional, or iterative execution [11]. The increasing complexity of the processes that underlie most human activities nowadays motivated the research on automatic process mining [13] and managing, requiring very powerful tools and representation formalisms. As a further constraint, in some cases the discovered information must be human-readable, so that human experts can validate it before its usage and/or use it to improve their insight and understanding of the domain and of the process itself. *Declarative* Process Mining [10] relies on formal logic to represent process-related information, and thus is a good candidate to support human readability of the learned models. The WoMan framework was shown in [4, 3] to be able to handle efficiently and effectively very complex processes where other state-of-the-art systems fail, especially due to short loops, duplicate activities and large concurrency (4-5 activities).

The objective of this paper is to stress WoMan and check whether it is able to handle even much larger concurrency (in the order of dozen activities), this way confirming its power. So, here we are not interested in introducing new features for handling this specific challenge, nor in assessing the superiority of declarative approaches to process mining versus more classical ones, nor in directly comparing WoMan to other systems. We pursue our objective considering the domain of chess playing, that we suggest can be seen as a (complex) process[1].

[1] We stress the fact that WoMan is a general framework, so it does not need any specific tailoring for handling this new domain.

Recent work on this topic investigated the use of Deep Learning [6] and Neural Networks [9], but the proposed approaches are sub-symbolic (i.e., models are not human readable) and require huge amounts of examples. WoMan overcomes both these shortcomings, working in the First-Order Logic (FOL) setting, since the various kinds of relationships involved in chess playing are fundamental to correctly grasp high-level information about it. So, a secondary objective of this paper is to check whether the learned information provides useful hints to humans about chess playing, using both qualitative and quantitative evaluations.

A *workflow* is a formal specification of a process [11]. A *case* is a particular execution of actions compliant to a given workflow. Case *traces* consist of lists of events associated to *steps* (time points). Several traces may be collected in *logs* [12]. A *task* is a generic piece of work. An *activity* is the actual execution of a task by a *resource* (an agent that can carry it out).

In our vision, playing a chess match corresponds to enacting a process. We consider a task as the occupation of a specific square of the chessboard by a specific kind of piece (e.g., "black rook in a8"), and the involved resources as the two players: 'white' and 'black'. Matches are initialized by starting 32 activities corresponding to the initial positions of all pieces on the chessboard. Each move terminates some activities (i.e., removes pieces from the squares they occupy) and starts new activities (i.e., occupies some squares by pieces). Some features of this setting represent hard problems for most process mining approaches in the literature: high parallelism (the number of concurrent activities during the match is beyond the reach of many current process mining systems [4]), short and nested loops (a piece, after a number of moves, going back on a square that it had occupied in the past), optional activities (a given move may or may not take an opponent's piece) and duplicate tasks (a piece may occupy the same square at different stages of the match, and thus in different contexts). So, it may represent a tough testbed to evaluate the process mining state-of-the-art.

This paper is organized as follows. In the next section we briefly recall the formalism and architecture of WoMan. Then, we explain how chess playing concepts can be transposed into a process-oriented representation, and we report experimental results showing that WoMan is able to learn this complex kind of process and to discover interesting information in that domain. Finally, in the last section, we draw some conclusions and outline future work issues.

2 Advanced Process Mining with WoMan

The WoMan framework [4, 3] introduced some important novelties and peculiarities in the process mining and management landscape. A fundamental one is the pervasive use of FOL as a representation formalism, that allows to describe contextual information using relationships. In particular, it works in the Logic Programming fragment of FOL [7], and thus it can be considered an Inductive Logic Programming (ILP) system [8]. Due to space limitations, the interested reader is referred to [4, 3] for a detailed presentation of WoMan's architecture,

features and representation formalism. Here, we will briefly and intuitively recall the notions that will be useful to understand the proposed application.

2.1 Representation

Following the foundational literature [1, 5], WoMan takes as input trace elements consisting of 7-tuples, represented as logic atoms of the form:

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

where T is the event timestamp, E is the type of the event (one of 'begin_process', 'end_process', 'begin_activity', 'end_activity'), W is the name of the workflow the process refers to, P is a unique identifier for each case, A is the name of the activity, O is the progressive number of occurrence of that activity in that case, and R is an optional field can be used to specify the resource that is carrying out the activity (it was not present in [1, 5]).

A model describes the structure of a workflow using the following elements:

tasks : the kinds of activities that are allowed in the process;
transitions : the allowed connections between activities (also specifying the involved resources).

Transitions carry all the information about the flow of activities during process execution. A transition $t : I \Rightarrow O$, where I and O are multisets of tasks, is enabled if all input tasks in I are active; it occurs when, after stopping (in any order) the concurrent execution of all tasks in I, the concurrent execution of all output tasks in O is started (again, in any order). For analogy with the notions of 'token' and 'marking' in Petri Nets, during a process enactment we call a *token* an activity that has terminated and can be used to fire a transition, and a *marking* the set of current tokens. Both tasks and transitions are associated to the multiset C of training cases in which they occurred (a multiset because a task or transition may occur several times in the same case). It can be exploited both for guiding the conformance check of new cases and for computing statistics on the use of tasks and transitions. In particular, it allows us to compute the probability of occurrence of a task/transition in a model learned from n training cases as the relative frequency $|C|/n$.

As shown in [4, 3], this representation formalism is more powerful than Petri or Workflow Nets [13], that are the current standard in Process Mining. The increased power of WoMan's representation formalism for workflow models raises some issues that must be tackled. In Petri Nets, since a single graph is used to represent the allowed flow(s) of activities in a process, at any moment in time during a process enactment, the supervisor knows which tokens are available in which places, and thus he may know exactly which tasks are enabled. So, the prediction of the next activities that may be carried out is quite obvious, and checking the compliance of a new activity with the model means just checking that the associated task is in the set of enabled ones. Conversely, in WoMan the activity flow model is split into several 'transitions', and different transitions

may share some input and output activities, which allows them to be composed in different ways with each other. 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. Such an ambiguity can be resolved only at a later time. Let us see this through an example. Suppose that our model includes, among others, the following transitions:

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

and that the current marking (i.e., the set of the latest activities that were terminated in the current process enactment, without starting any activity after their termination) is $\{x, y, z\}$. Now, suppose that activity a is started. It might indicate that either transition t_1 or transition t_2 have been fired. Also, if an activity d is currently being executed due to transition t_3, the current execution of a might correspond to the other output activity of transition t_3, which we are still waiting to happen to complete that transition. 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, as we will show later.

2.2 Architecture

Several modules are included in WoMan to perform the different tasks involved in learning and managing process models. **WIND** (Workflow INDucer) allows one to learn or refine a process model according to a case, after the case events are completely acquired. Differently from all previous approaches in the literature, WoMan's learning module 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 is a significant advance with respect to the state-of-the-art, because continuous adaptation of the learned model to the actual practice can be carried out efficiently, effectively and transparently to the users [3].

PAT (Process Analysis Tool) allows one to perform several kinds of analysis on the learned model, such as identifying the most frequently used model components, comparing the frequent components among different models, comparing the structure of different models, etc..

The supervision module allows one to check whether new cases are compliant with a given model. **WEST** (Workflow Enactment Supervisor and Trainer) takes the case events as long as they are available, and returns information about their compliance with the currently available model for the process they refer to. The output for each event can be 'ok', 'error' (e.g., when closing activities that had never begun, or ending the execution while activities are still running), or a set of warnings denoting different kinds of deviations from the model (e.g., unexpected task or transition, preconditions not fulfilled, unexpected resource running a given activity, etc.).

Now, as we have pointed out in the previous section, given an intermediate status of the process enactment and a new activity that is started, there may be different combinations of transitions that are compliant with the new activity, and one may not know which is the correct one until a later time. Thus, all the corresponding alternate evolutions of the status must be carried on by the system. Considering again the previous example, each of the three proposed options would change in a different way the status of the process, as follows:

t_1 : firing this transition would consume x, yielding the new marking $\{y, z\}$ and causing the system to wait for a later activation of b;
t_2 : firing this transition would consume x and y, yielding the new marking $\{z\}$ and causing the completion of transition t_2;
t_3 : firing this transition would not consume any element in the marking, but would cause the completion of transition t_3.

When the next event is considered, each of these evolutions is a possible status of the process enactment. On the one hand, it poses again the same ambiguity issues; on the other hand, it may point out that some current alternate statuses were wrong. 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.

Note also that each alternative may be compliant with a different set of training cases, and may rise different warnings (in the previous example, one option might be fully compliant with the model, another might rise a warning for task preconditions not fulfilled, and the other might rise a warning for an unexpected agent running that activity). WEST takes note of the warnings for each alternative and carries them on, because they might reflect secondary deviations from the model that one is willing to accept. Wrong alternatives will be removed when they will be found out to be inconsistent with later events in the process enactment. So, the question arises about how to represent each alternative status. As suggested by the previous example, we may see each status as a 5-tuple

$$\langle M, R, C, T, W \rangle$$

recording the following information:

M the marking, i.e., the set of terminated activities that have not been used yet to fire a transition, each associated with the agent that carried it out and to the transition in which it was involved as an output activity;
R the set of activities that are 'ready' to start, i.e., the output activities of transitions that have been fired in the status, and that the system is waiting for in order to complete those transitions;
C the set of training cases that are compliant with that status;
T the set of (hypothesized) transitions that have been fired to reach that status;
W the set of warnings raised by the various events that led to that status.

While in supervision mode, the prediction modules allow one to foresee which activities the user is likely to perform next, or to understand which process is being carried out among a given set of candidates. We recall that, due to the discussed set of alternate statuses that are compliant with the activities carried out at any moment, differently from Petri Nets it is not obvious to determine which are the next activities that will be carried out. Indeed, any status might be associated to different expected evolutions. The good news is that, having several alternate statuses, we can compute statistics on the expected activities in the different statuses, and use these statistics to determine a ranking of those that most likely will be carried out next.

Specifically, **SNAP** (Suggester of Next Action in Process) hypothesizes which are the possible/appropriate next activities that can be expected given the current intermediate status of a process execution, ranked by confidence. Confidence here is not to be interpreted in the mathematical sense. It is determined based on a heuristic combination of several parameters associated with the possible alternate process statuses that are compliant with the current partial process execution. Specifically, the activities that can be carried out next in a given status are those included in the *Ready* component of that status, or those belonging to the output set of transitions that are enabled by the *Marking* component of that status. The status parameters used for the predictions are the following:

1. frequency of activities across the various statuses (activities that appear in more statuses are more likely to be carried out next);
2. number of cases with which each status is compliant (activities expected in the statuses supported by more training cases are more likely to be carried out next);
3. number of warnings raised by the status (activities expected in statuses that raised less warnings are more likely to be carried out next);
4. confidence of the tasks and transitions as computed by the multiset of cases supporting them in the model (activities supported by more confidence, or belonging to transitions that are associated to more confidence, are more likely to be carried out next).

Finally, given a case of an unknown workflow, **WoGue** (Workflow Guesser) returns a ranking (by confidence) of a set of candidate process models. Again, this prediction is based on the possible alternate statuses identified by WEST when applying the events of the current process enactment to the candidate models. In this case, the candidate models are ranked by decreasing performance of their 'best' status, i.e. the status reporting best performance in (one or a combination of) the above parameters.

3 Evaluation: Chess Playing as Process Execution

For our experiments we downloaded 400 matches (200 played by Karpov and 200 played by Kasparov) from the Website of the Italian Chess Federation (`http://scacchi.qnet.it`), and translated them from PGN (Portable Game

Table 1. Dataset statistics

	# matches				events		tasks		runtime (sec)	
Player	total	white	black	draw	total	avg	total	avg	total	avg
Karpov	200	79	48	73	45088	225.44	22344	111.72	43.192	0.215
Kasparov	200	79	39	82	45244	226.22	22422	112.11	41.192	0.205
Total	400	158	87	155	90332	225.83	44366	110.915	94.4	0.236

Notation), an open source format representing the moves in algebraic notation, to the input format of WoMan. Each log entry bears the following information:

T a progressive number indicating the event timestamp;
E one of the allowed event types:
  begin_of_process the start of a match;
  begin_of_activity a certain piece is placed in a certain square;
  end_of_activity a certain piece is removed from a certain square;
  end_of_process the end of a match.
W the name of the process model the entry is referred to;
P a unique match identifier, obtained by concatenation of the following data: name of white player, name of black player, place and date of the match;
A the name of the activity, encoded as 4 characters denoting, respectively: chessboard file, chessboard rank, piece (in the following we used the Italian initials: p = pawn, t = rook, a = bishop, c = knight, d = queen, r = king), and player (b = white, n = black);
O the progressive number of occurrence of A in P;
R the player (*white* or *black*) responsible for the beginning or end of activity.

As regards the processes of interest, we considered three processes, corresponding to the possible match outcomes: *white* wins, *black* wins, or *draw*. Then we used the following WoMan functionality. First we used WEST, to check compliance of each training case with the current models, in combination with WIND, to learn and refine the models after the compliance check of each case. On the final learned models, we used PAT, to see whether interesting information was discovered and 'compiled' in the models. Finally, we used SNAP and WEST to check whether correct predictions could be obtained using the learned models, in this way indirectly evaluating the quality of the discovered information.

Table 1 reports statistics on the dataset and on the learning runtime of WoMan. The number of matches for the three processes is similar for the two players. Note that the average (*avg*) number of events and tasks are nearly identical, suggesting that it does not depend on the specific players. Also the average runtime needed to learn the models is almost identical for the two subsets, suggesting it only depends on the number of cases. For the whole dataset, including twice as much cases as the single subsets, average runtime is still comparable to those of the subsets, which confirms that WoMan has an acceptable time complexity (linear or loglinear) in the number of processed cases. Less than a quarter of a second is needed on average to incorporate a new match in the available model, which means, considering that each match involves nearly 226 events on

Fig. 1. Incremental learning behavior

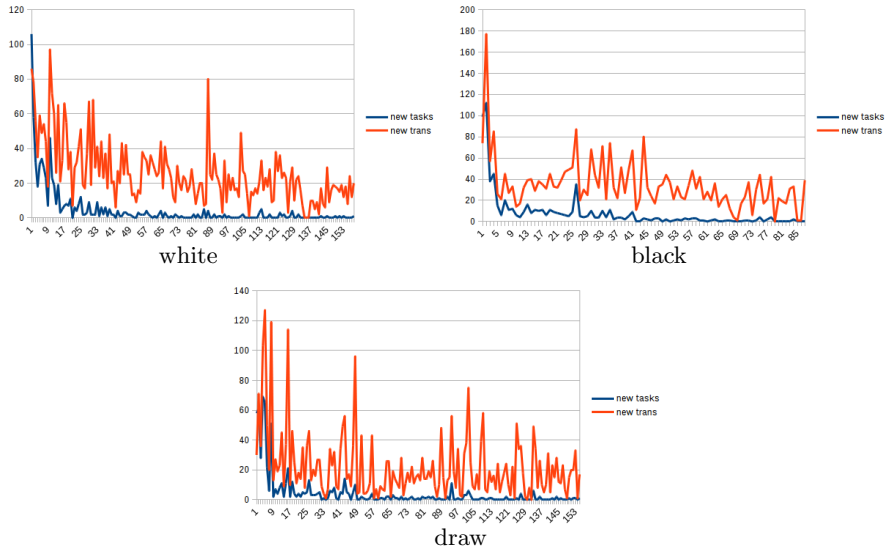
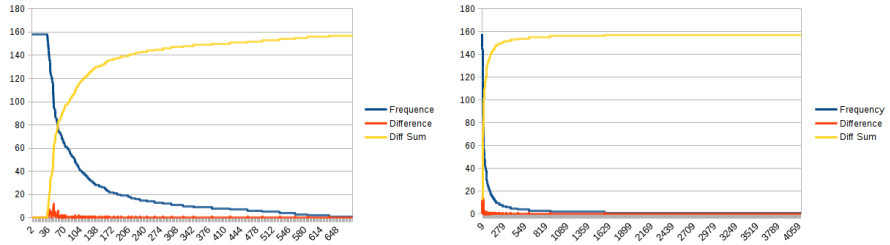


Table 2. Model statistics

	Karpov			Kasparov			Karpov + Kasparov		
	white	black	draw	white	black	draw	white	black	draw
# matches	79	48	73	79	39	82	158	87	155
# tasks	629	624	588	635	567	615	681	663	658
(avg)	3.14	3.12	2.94	3.17	2.83	3.07	1.7	1.65	1.39
# transitions	2665	2055	1949	2445	1575	2318	4083	3006	3434
(avg)	13.32	10.27	9.74	12.22	7.87	11.59	10.20	7.51	8.58

average, that each event is processed in about 1 msec. We may conclude that WoMan has proven its ability to learn such complex kinds of workflows. Figure 1 shows the dynamic learning behavior. The graphics report the number of new tasks (blue line) and transitions (red line) that were found in each new training case, compared to those already learned from all previous cases. The peaks, especially for tasks, become progressively lower and sparser, which confirms that the system is actually converging on a subset of very relevant tasks and transitions.

Table 2 shows that the number of tasks and transitions is comparable in the models learned from the two training subsets. The number of tasks is almost the same also in the overall dataset, which was expected because the number of all possible piece positions is small (750) compared to the number of tasks in the training sets. Nevertheless, the models actually use only up to 90% of all possible tasks, suggesting that some piece positions are undesirable. Conversely, the number of different transitions from the subsets to the overall dataset grows, as expected, but the growth is less than linear, even if they are much less than



(a) (b)

Fig. 2. Statistics on tasks and transitions frequency for the *white* process

Table 3. Most frequent tasks for the different processes

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
w	d4pb	f6cn	f3cb	f1tb	e4pb	g1rb	f8tn	g8rn	c3cb	c5pn	c4pb	e6pn	d5pn	c6cn	d6pn	d1tb	d7cn	f4pb	g6pn	e5pn
b	f6cn	f3cb	d4pb	g8rn	f8tn	c3cb	g1rb	f1tb	e4pb	c5pn	c4pb	d5pn	c6cn	e6pn	e5pn	d1tb	d6pn	d4pn	d5pb	b5pn
d	f6cn	d4pb	f8tn	f3cb	g8rn	f1tb	g1rb	c3cb	c5pn	c4pb	e6pn	e7an	e4pb	d5pn	c6cn	d5pb	d6pn	e5pn	g3pb	d4pn

all possible ones. This may indicate that only a small portion of moves is really relevant to players. We also note that the number of different moves is neatly larger when the white wins than when the black wins.

Let us now focus on the frequency of tasks and transitions. The blue line in Figure 2 plots the number of cases in which the different tasks (a) and transitions (b) occur in the *white* model, by decreasing frequency. The same information is shown also as difference between adjacent values (red line), and as cumulative difference in occurrences (yellow line). The initial plateau in (a) is given by the initial position of pieces on the chessboard, that of course occurs in all training cases. The shape is similar for all processes, showing a clear drop in frequency very early (i.e., toward the left) along the x-axis, that is evident also in the series of close peaks in the red line. It suggests which are the most frequent items on which further analysis can be carried out. Table 3 shows the 20 most frequent (non-initial) tasks for each process (in decreasing order from left to right). Very interestingly, we find in the top positions the tasks that correspond to well-known openings widely acknowledged in the chess literature. E.g., in all processes *d4pb* (white pawn in d4) is in the top 3 positions, *f3cb* (white knight in f3) is in the top 4 positions, and *e4pb* (white pawn in e4) is in the top 13 positions.

Since many tasks and transitions appear in all processes, the question arises about which ones, if any, are more characteristic or discriminant for the final outcomes of a match (i.e., for each process). We took the set of k most frequent tasks for each process, for different values of k, and computed the differences among such sets. The lower k, the more characteristic the items in such a difference are expected to be for a given process. E.g., considering the top 10 tasks in Table 3, *e4pb* (white pawn in e4) is present in *white* and *black*, but not in *draw*, suggesting that it may be discriminant for the latter. Conversely, *c4pb* (white pawn in c4) somehow characterizes *draw*, because it is in its 10 most frequent tasks, but not in the top 10 of the others. Clearly, there is an obvious skew of

Table 4. Characteristic and discriminant tasks for the different processes

White	never	a4cn, a5an, b2pn, b2rb, b4cb, b7cb, c1cn, c8ab, d2pn, e2cn, e8cb, f1cn, f1dn, f2cn, f2dn, f3tn, f4an, g1dn, g1tn, g2dn, g2tb, h1dn, h2an, h3db, h7ab
	always	a1cn, a7cn, a7pb, a7rn, a8cn, a8rn, b2rn, b7pb, b7rb, b8ab, c2pn, c2rn, c5rb, d7rb, d8cb, e7pb, e7rb, f7ab, f7rb, g1ab, g1rn, g2pn, g2rn, g3rn, g8an, h1cb, h2db, h2pn, h2tb, h3rn, h5rb, h7cb, h8cn, h8db, h8dn
Black	never	a1db, a2cb, a2cn, a4rn, a5rb, a5rn, a6ab, a6db, a6pb, a6rb, a7ab, a7tn, b1db, b3tb, b4tn, b5rb, b6rb, b7cn, b8an, b8rn, c4rb, c6rb, c8cn, c8db, d1an, d2an, d3rn, d3tn, d6rb, d7ab, d8rn, d8tb, e1dn, e2an, e4rb, e5rb, e6db, e8ab, e8dn, f6cn, f6db, f6rb, f6tb, f7cn, f7dn, f7pb, f7tn, f8cb, f8cn, g2cb, g3an, g4cb, g4tb, g5an, g5cn, g5db, g6cb, g6db, g7dn, g7pb, h2ab, h2dn, h3pn, h4cn, h4rn, h4tb, h5db, h5rn, h6cb, h6db, h6dn, h6tn, h7db, h7dn, h7tn
	always	a1ab, a1cb, a1rb, a1tb, a2rb, a3cn, a3rb, c1rn, c3rn, d2rn, e1cn, f3rn, f4rn, g1an, g5rb, g8ab, h1db
Draw	never	a2db, a4rb, a5ab, a6cb, b1an, b3dn, b4rb, b6db, b6pb, b8cb, c7pb, c7rn, d1cn, d4rn, e2pn, e3pn, e6ab, f2pn, f3cn, f4tn, f5rb, f5rn, f5tb, f8ab, g2tn, h1tn, h2tn, h3tn, h4tn, h5ab, h8an, h8tb
	always	a1an, a2an, a6rn, a8ab, b3an, b4rn, b5cn, b5rn, c1an, c1cb, c4rn, c5rn, d8cn, e1ab, e1an, e2rn, e3rn, f1an, g1db, g8db, h4an, h7pb

frequencies in favor of initial moves, where the possibilities are more limited, and thus we expect to find initial moves of the pieces for low values of k, while for higher values of k we expect that all kinds of processes have exploited all possible initial moves, and thus the characterizing or discriminant items are more related to the middle of the match. Table 4 shows, for each process, all the tasks that *never* occurred (but that occurred in the other processes), and all those that *always* occurred (but never occurred in the other processes).

These considerations are confirmed for transitions. For $k = 20$, characterizing moves for white-winning matches are *[f2pb]-[f4pb]* (white pawn moving two squares from its initial position f2) and *[b8cn]-[d7cn]* (black knight moving from b8 to d7). The latter move may be interesting, because it cannot be done before the pawn initially placed in d7 moves away, and thus there are less chances that it happens frequently in the initial moves.

As regards the prediction of next activities, we report in Table 5 some statistics obtained from an 80%-20% training-test set random split procedure for the different processes. Column 'predictions' reports in how many cases SNAP returned a prediction (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). Among the cases in which WoMan makes a prediction, column 'correct' reports in how many of those predictions the correct activity (i.e., the activity that is actually carried out next) is present, and column 'Ranking' reports how close it is to the first element of the ranking (1.0 meaning it is the first in the ranking, possibly with other activities, and

Table 5. Prediction statistics

	Activity				Process				
	Predictions	Correct	Ranking	Quality	Acc	Avg	F	A	L
black	0.42	0.98	1.0	0.41	0.50	2.0	0.25	14.62	77.12
white	0.53	0.98	1.0	0.52	0.37	2.26	0.25	18.75	69.17
draw	0.69	0.97	1.0	0.67	0.73	1.54	0.85	32.35	73.5
overall	0.55	0.98	1.0	0.53	0.53	1.93	0.45	21.91	73.26

0.0 meaning it is the last in the ranking). The 'Quality' index is a mix of these values, obtained as $Predictions \cdot Correct \cdot Ranking$. It ranges from 0.0, meaning that predictions are completely unreliable, to 1.0, meaning that WoMan always makes a correct prediction. Interestingly, when a prediction is made it is almost certainly correct, and contains the next actual activity at the top of the ranking. The number of predictions made is also interesting, given that in more than half of the match WoMan is able to suggest which will be the next action in the game. This is quite interesting, also compared to the state-of-the-art [6, 9], and indicates that significant information about chess was discovered in the models. The worse performance is on 'black', which is the one with less training cases.

Finally, Table 5 reports also the performance on the process prediction. *Avg* is the average position of the correct prediction in the ranking; *Acc* reports the accuracy, computed by normalizing *Avg* (1 meaning the correct process is the first in the ranking, 0 meaning it is the last). Columns F, A, L report respectively, on average, at what point during a case the following events occur: the correct process becomes the *F*irst in the ranking for the first time (possibly sharing the first position with others); the correct process becomes *A*lone the first in the ranking for the first time; the correct process is not the first in the ranking for the *L*ast time. Each process performs best on a different parameter: 'black' on A, 'white' on L, 'draw' on Accuracy. Interestingly, while the overall accuracy is not outstanding, the correct process appears very early at the top of the ranking, and after less than 3/4 of the process enactment on average the correct process is definitively assessed.

4 Conclusions and Future Work

The main objective of this paper was checking whether, and to what extent, advanced process mining techniques, and specifically the WoMan framework for process mining and management, can support efficient and effective knowledge discovery in complex domains. For this purpose, we focused on chess playing, and cast it as a process. The experimental outcomes showed that, albeit further work is to be carried out, satisfactory initial results have been obtained for all objectives. From the process mining perspective, WoMan proved able to learn chess models effectively and efficiently. From the chess perspective, the learned models discovered interesting features of the game, including (fragments of) some well-known notions and techniques in the chess literature.

As a future work, we will investigate the other features of WoMan, to check whether additional useful information can be discovered (e.g., it may learn preconditions that correspond to the rules of the game or to relevant game strategies). We also plan to find other domains that provide complex processes from which trying to discover useful information.

Acknowledgments

This work was partially funded by the Italian PON 2007-2013 project PON02_00563_3489339 'Puglia@Service'.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, 1998.
2. J.E. Cook and A.L. Wolf. Discovering models of software processes from event-based data. Technical Report CU-CS-819-96, Department of Computer Science, University of Colorado, 1996.
3. S. Ferilli. Woman: Logic-based workflow learning and management. *IEEE Transaction on Systems, Man and Cybernetics: Systems*, 44:744–756, 2014.
4. S. Ferilli and F. Esposito. A logic framework for incremental learning of process models. *Fundamenta Informaticae*, 128:413–443, 2013.
5. J. Herbst and D. Karagiannis. 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*, pages 52–57, 1999.
6. M. Lai. Giraffe: Using deep reinforcement learning to play chess. *CoRR*, abs/1509.01549, 2015.
7. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2 edition, 1987.
8. S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.
9. B. Oshri and N. Khandwala. Predicting moves in chess using convolutional neural networks. In *Stanford University Course Project Reports – CS231n: Convolutional Neural Networks for Visual Recognition*. Winter 2016.
10. M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 international conference on Business Process Management Workshops*, BPM'06, pages 169–180. Springer-Verlag, 2006.
11. W.M.P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8:21–66, 1998.
12. W.M.P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16:1128–1142, 2004.
13. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data. In V. Hoste and G. De Pauw, editors, *Proceedings of the 11th Dutch-Belgian Conference of Machine Learning (Benelearn 2001)*, pages 93–100, 2001.