Completion Time and Next Activity Prediction of Processes Using Sequential Pattern Mining

Michelangelo Ceci, Pasqua Fabiana Lanotte, Fabio Fumarola,
Dario Pietro Cavallo, and Donato Malerba

Dipartimento di Informatica, University of Bari "Aldo Moro", Bari, Italy
{michelangelo.ceci,pasquafabiana.lanotte,fabio.fumarola,
donato.malerba}@uniba.it, dario.pt.cavallo@gmail.com

Abstract. Process mining is a research discipline that aims to discover, monitor and improve real processing using event logs. In this paper we describe a novel approach that (i) identifies partial process models by exploiting sequential pattern mining and (ii) uses the additional information about the activities matching a partial process model to train nested prediction models from event logs. Models can be used to predict the next activity and completion time of a new (running) process instance. We compare our approach with a model based on Transition Systems implemented in the ProM5 Suite and show that the attributes in the event log can improve the accuracy of the model without decreasing performances. The experimental results show how our algorithm improves of a large margin ProM5 in predicting the completion time of a process, while it presents competitive results for next activity prediction.

1 Introduction

Today, many organizations store event data from their enterprise information system in structured forms such as *event logs*. Examples of such logs are audit trails of workflow management systems, transaction logs from enterprise resource planning systems, electronic patient records, etc.. Here, the goal is not to just collect as much data as possible, but to extract valuable knowledge that can be used to compete with other organizations in terms of efficiency, speed and services. These issues are taken into account in *Process Mining*, whose goal is to discover, monitor and improve processes by providing techniques and tools to extract knowledge from event logs. In typical application scenarios, it is assumed that events are available and each event: (i) refers to an *activity* (i.e., a well-defined step in some process), (ii) is related to a *case* (i.e., a process instance), (iii) can have a *performer* (the actor executing or initiating the activity), and (iv) is executed at a given *timestamp*. Moreover, an event can carry additional process-specific attributes (e.g. the cost associated to the event, the place where the event is performed).

Event logs such as the one shown in Table 1 are used as the starting point for mining. As described in [12], we distinguish four different analyses: (1) *Discovery*, (2) *Conformance*, (3) *Enhancement* and (4) *Operational Support*. In *Discovery*,

S. Džeroski et al. (Eds.): DS 2014, LNAI 8777, pp. 49–61, 2014.

a process model is discovered based on event logs [7,14]. For example, the α-algorithm [14] mines a process model represented as a *Petri-Net* [4,2] from event logs. In *Conformance analysis*, an existing process model is compared with event logs to check and analyze discrepancies between the model and the log. Viceversa, the idea behind *Enhancement* is to extend or improve an existing process model using information about the actual process recorded in some event logs. Types of enhancement can be *extension*, i.e., adding new perspectives to a process model by cross-correlating it with a log, or *repair*, i.e., modify an discovered model to better reflect reality. It is noteworthy that in all the analysis considered above, it is assumed that process mining is done *offline*. Processes are analyzed thereafter to evaluate how they can be improved or extended. On the contrary, *Operational Support* techniques are used in *online* settings. Given a process model built over some event logs and a partial trace, operational support techniques can be used for detecting deviation at runtime (**Detect**), predicting the remaining processing time (**Predict**) and recommending the next activity (**Recommend**).

At the best of our knowledge, classical algorithms presented in the literature for operational support, (1) build a process model in form of Transition Systems [11] or Petri-Nets [4,2], (2) re-analyze the log to extend the model with temporal information and aggregated statistics [13], and finally, (3) learn a regression or a classification model to support prediction and recommendation activities. However, as noted in [5], these operational support methods naturally fit cases where processes are very well-structured (i.e. perfectly matching some predefined schema), for real-life logs they suffer of problems related to "incompleteness" (i.e. the model represents only a small fraction of the possible behavior due to the large number of alternatives), "noise" (i.e., logs containing exceptional/infrequent activities that should not be incorporated in the model), "overfitting" and "under-fitting", thereby resulting in a spaghetti-like model, which is rather useless in practice. Other approaches, such as computational intelligence systems [7], which overcome these problems, tend to be inefficient and, thus, have problems to scale in case of a huge amount of activities which are correlated each other (by means of precedence/causality dependencies).

In this paper we present a novel approach for operational support which deals with the problems presented before, that is, "incompleteness", "robustness to noise" and "overfitting". The solution we propose aims at identifying partial process models to be used for training predictive models. In our approach, two types of predictive models are inferred: for the prediction of the next activity and for the estimation of the completion time. In details, we identify frequent partial processes in form of frequent activity sequences. These sequences are extracted by adapting an efficient frequent pattern mining algorithm and are represented in form of sequence trees. Afterwards, we associate at each node of the tree a specific prediction model that takes into account, in addition to classical attributes (such as the performer of each activity), also additional attributes such as the cost associated to the event or the place where the event is performed. We call this last prediction model "nested". While the sequence mining algorithm allows us to deal with incompleteness, robustness to noise and overfitting by removing

Table 1. An example of event log

CID	Act	Time	Perf	X	Y	CID	Act	Time	Perf	X	Y
1	A	0	p1	x1	y1	4	C	22	p1	-	-
1	B	6	p1	-	-	4	D	28	p1	-	-
1	C	12	p1	-	-	5	A	18	p1	x1	y2
1	D	18	p1	-	-	5	C	22	p2	-	-
2	A	10	p2	x1	y1	5	B	26	p1	-	-
2	C	14	p2	-	-	5	D	32	p2	-	-
2	B	26	p2	-	-	6	A	19	p1	x2	y2
2	D	36	p2	-	-	6	E	28	p3	-	-
3	A	12	p3	x2	y2	6	D	59	p2	-	-
3	E	22	p3	-	-	7	A	20	p1	x2	y1
3	D	56	p3	-	-	7	C	25	p3	-	-
4	A	15	p1	x1	y1	7	B	36	p3	-	-
4	B	19	p1	-	-	7	D	44	p2	-	-

unfrequent behaviors, the nested models guarantee some flexibility. In fact, it is possible to *i)* plug-in any classification/regression learning algorithm and *ii)* enable a different representation of the data, one for each node of the trees. Our solution has its inspiration in works which face with the associative classification task [3], where descriptive data mining techniques are exploited for predictive purposes using a hybrid data mining approach.

The paper is organized as follows: in the next section we describe the proposed approach. Section 3 is devoted to present the empirical evaluation of the proposed solution. Finally, Section 4 concludes the paper and draws some future work.

2 Methodology

This section describes our two-stepped online operational support approach.

First Phase: Process Discovery
In this phase, we look at a (partial) process as a sequence of activities and we apply a sequential pattern mining algorithm in order to generate a partial process model. This model allows us to represent both complete and partial traces which are found frequent by the algorithm. The algorithm we adopt in this phase is FAST [9] which guarantees low computational costs and allows us to represent frequent sequences in a compact way by means of *sequence trees*. FAST, by focusing only on frequent sequences, leads to predictive models (see Section 16) which are robust to noise and do not suffer from overfitting problems. Moreover, since FAST is able to extract frequent non-contiguous (and partial) sequences of activities, we are also able to deal with incompleteness problems.

In this work we extend FAST by allowing it to also extract: *1)* only contiguous (partial) sequences and *2)* only contiguous (partial) sequences that represent only processes since their beginning. Obviously, these extensions generate smaller sequence trees, improve robustness to noise, reduce overfitting, but increase problems related to incompleteness. All these aspects are due to a smaller number of sequence patterns and, thus, to less specialized models. It is noteworthy that the idea of reducing the size of the process model is not new and in [13]

the authors convert sequences into sets and multi-sets. However, such approach results in loosing the exact order of events and the number of the occurrences.

In our approach, an event log is represented as a sequence database *(SDB)*, that is, a set of tuples $\langle CID, S\rangle$, where CID is a case id and S is a sequence of ordered set of events (i.e. set of activities). Figure 1a shows the sequence database extracted from the event log reported in Table 1. For instance, in this figure, the cases 1 and 4 are composed by the sequence of activities A, B, C, D.

To formally describe the task solved in this phase, we give some definitions.

Definition 1 (Sub/Super-Sequence). *A sequence $\alpha = \langle a_1, a_2, \ldots, a_k\rangle$ is called a sub-sequence of another sequence $\beta = \langle b_1, b_2, \ldots, b_m\rangle$, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j_1 \leq j_2 \leq \ldots \leq j_k \leq m$ such that $a_1 = b_{j_1}, a_2 = b_{j_2}, \ldots, a_k = b_{j_k}$. For example, if $\alpha = \langle A, B\rangle$ and $\beta = \langle A, C, B\rangle$, where A, C and B are events, then α is sub-sequence of β and β is a super-sequence of α.*

Definition 2 (Frequent sequences). *Let $\alpha = \langle a_1, a_2, \ldots, a_k\rangle$ be a sequence of activities, SDB be a sequence database and $minsup$ a user-defined threshold. α is frequent if its support $\sigma(\alpha, SDB)$ (i.e. the number of sequences in SDB which are super-sequences of α) is greater than $minsup$.*

Definition 3 (Contiguous frequent sequences). *Let $\alpha = \langle a_1, a_2, \ldots, a_k\rangle$ be a frequent sequence in a sequence database SDB, according to a user-defined threshold $minsup$. We define the* contiguous support *of α (denoted as $\sigma_{cs}(\alpha, SDB)$) as the number of sequences in SDB containing α such that, for each $i = 1 \ldots k-1$, the activity a_{i+1} is observed immediately after the activity a_i. If $\sigma_{cs}(\alpha, SDB) \geq minsup$, then α is a* contiguous sequence*.*

Definition 4 (Opening frequent sequence). *Let $\alpha = \langle a_1, a_2, \ldots, a_k\rangle$ be a contiguous frequent sequence in a sequence database SDB and $minsup$ a user-defined threshold. We define the* opening support *of α (denoted $\sigma_{os}(\alpha, SDB)$) as the number of sequences in SDB containing α, and having a_1 in the first position. If $\sigma_{os}(\alpha, SDB) \geq minsup$, then α is an* opening sequence*.*

Example 1. Given the sequence database SDB in Figure 1a and $minsup = 1$, we analyze the sequence $\alpha = \langle A, B\rangle$. α is frequent because it is present in the tuples with $CID = \{1, 2, 4, 5, 7\}$. Now we check if α can be marked as *contiguous frequent sequence*. The activities A, B appear contiguously only in the tuples with $CID = \{1, 4\}$ (i.e. $\sigma_{cs}(\alpha, SDB) = 2$). Since $\sigma_{cs}(\alpha, SDB) \geq minsup$, α is marked as contiguous frequent sequence. Finally, we check if α can be marked as *opening frequent sequence*. In this case, we have to count how many times the first activity of α (i.e. A) is observed in first position in the SDB and its next activity is B. Since this happens for $CID = \{1, 4\}$ (i.e. $\sigma_{os}(\alpha, SDB) = 2$), α is an opening frequent sequence.

The above definitions allow us to push constraints in the patterns used to build partial process models. Methodologically, we face the following task: given a sequence database SDB, where each sequence represents a sequence of events and given a user-specified minimum support threshold $minsup$, the task of process

CID	Sequence
1	<A,B,C,D>
2	<A,C,B,D>
3	<A,E,D>
4	<A,B,C,D>
5	<A,C,B,D>
6	<A,E,D>
7	<A,C,B,D>

(a)

[2]
[3]
NULL
[2]
[3]
NULL
[3]

(b)

Fig. 1. (a) Sequence Database extracted from Table 1 and (b) VIL for $\langle A, B\rangle$

discovery is to find either: *i)* the frequent sequences in SDB, *ii)* the contiguous frequent sequences in SDB or *iii)* the opening sequences in SDB. In all the cases, sequences are expressed in the form of sequence trees. The original version of FAST can generate a sequence tree of activities in two phases. In the first phase all frequent activities are selected then, in the second phase, these activities are used to populate the first level of the sequence tree and to generate sequences with size greater than one. The mined sequence tree is characterized by the following properties: 1) each node in the tree corresponds to a sequence and the root corresponds to the null sequence (<>); 2) if a node corresponds to a sequence s, its children are generated by adding to s the last activity of its siblings. Only frequent children are stored in the tree. Figure 2 shows the sequence tree extracted by FAST from the database in Table 1a ($minsup = 1$).

To represent in optimized way the sequence dataset and to perform efficient support counting of activities and sequences, FAST uses a data structure called *vertical id-list* (VIL). In the following we give a brief definition of a VIL.

Definition 5 (Vertical Id-list). *Let SDB be a sequence database of size* n *(i.e. $|SDB| = n$), $S_j \in SDB$ its j-th sequence ($j \in \{1, 2, \ldots, n\}$), and α a sequence associated to a node of the tree, its* vertical id-list*, denoted as VIL_α, is a vector of size n, such that for each $j = 1, \ldots, n$*

$$VIL_\alpha[j] = \begin{cases} [posAct_{\alpha,1}, posAct_{\alpha,2}, \ldots, posAct_{\alpha,m}] & \text{if } S_j \text{ contains } \alpha \\ null & \text{otherwise} \end{cases}$$

where $posAct_{\alpha,i}$ is the end position of the i-th occurrence ($i \leq m$) of α in S_j.

Example 2. Figure 1b shows the VIL of the sequence $\alpha = \langle A, B\rangle$. Values in VIL_α represent the end position of the occurrences of the sequence α in the sequences of Figure 1a. In particular, the first element (list with only value 2) represents the position of the first occurrence of activity B, after the activity A (i.e. B is the last activity in α), in the first sequence S_1. The second element is (list with only value 3) the position of the first occurrence of B (after A) in the sequence S_2. The third element is null since α is not present in S_3. The other values are respectively list with only value 2 (for sequence S_4), list with only value 3 (for S_5), *null* (for S_6) and list with only value 3 (for S_7).

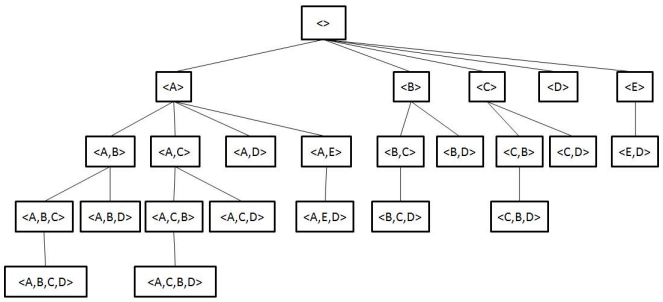

Fig. 2. Sequence tree learned with FAST

Given the sequence α and its sibling β FAST builds a new node γ and its VIL, using VIL_α and VIL_β. In particular, for each $j = 1, \ldots, n$, given: $\bullet VIL_\alpha[j] = [posAct_{\alpha,1}, \ldots, posAct_{\alpha,m_{\alpha,j}}]$,

- an index i (initialized to 1) on $VIL_\alpha[j]$,
- $VIL_\beta[j] = [posAct_{\beta,1}, \ldots, posAct_{\beta,m_{\beta,j}}]$,
- an index z (initialized to 1) on $VIL_\beta[j]$.

FAST checks whether $posAct_{\alpha,i} < posAct_{\beta,z}$. That is, the last activity of the first occurrence of α precedes the last activity of the first occurrence of β. If the condition is not satisfied, FAST increments z. This process is applied until either $posAct_{\alpha,i} < posAct_{\beta,z}$ is satisfied or the *null* value is found. In the first case FAST sets $VIL_\gamma[j] = [posAct_{\beta,z}, \ldots, posAct_{\beta,m_{\beta,j}}]$, otherwise $VIL_\gamma[j] = null$. The support of γ is then computed as: $\sigma(\gamma, SDB) = |\{j \mid VIL_\gamma[j] \neq null\}|$.

In our extension of FAST, we employ the VILs not only in the extraction of frequent sequences, but also in the extraction of contiguous and opening frequent sequences. In particular, the VIL structure is used to generate three different types of trees, representing frequent sequences (see Figure 2), contiguous frequent sequences (see Figure 3a) and opening sequences (see Figure 3b), respectively. It is interesting to note that this last type of sequence tree corresponds to a transition system model [12] obtained from the same event log, after removing unfrequent activities. These three different types of sequence trees are hereafter denoted as (proper) *sequence tree*, *contiguous tree* and *opening tree*. The contiguous tree is obtained by substituting, in the original implementation of FAST, the common definition of support with $\sigma_{cs}(n, SDB) = |\{j|vil = contiguous(n, T) \wedge vil[j] \neq NULL\}|$. In this formula, $contiguous(n, T)$ is the VIL returned by the application of the function described in Algorithm 1. This algorithm iteratively looks for possible holes in the sequences by bottom-up climbing the sequence tree. A hole is found when the condition at line 10 is not satisfied. Similarly, the opening tree is obtained by substituting, in FAST, the common definition of support with $\sigma_{os}(n, SDB) = |\{j|vil = contiguous(n, T) \wedge vil[j][1] = 1\}|$.

~~Sencond~~ Phase: Nested-Model Learning

Once the partial process model is learned, it is used to build a nested model for operational support. In particular, for each node α of the partial process model, the description of the processes, which contribute to the support of the

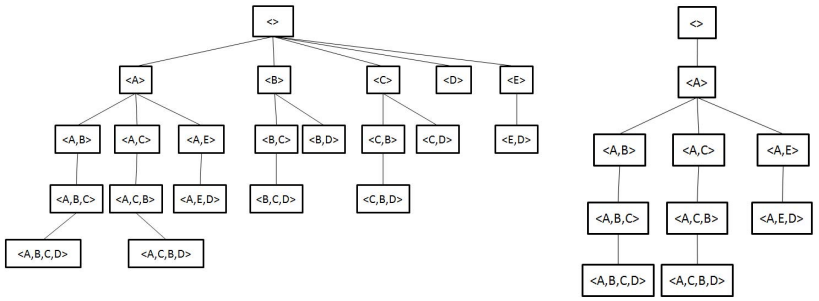

Fig. 3. (a) Contiguous and (b) Opening trees associated to the tree in Figure 2

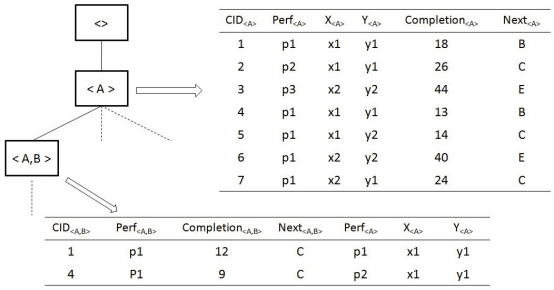

Fig. 4. Datasets associated with nodes $\langle A \rangle$ and $\langle A, B \rangle$

pattern expressed at that node, is used as a training set for a predictive learning algorithm. Learned prediction models are then used to predict the completion time and the next activity of a running process.

In the generation of the training set associated with each node α, all the informative attributes associated with each event in α, and available in an event log, can be used. Uninformative attributes, such as the CID, are removed, while two additional attributes, that is, *Completion time* and *Next Activity* can be created and populated for predictive purposes. Figure 4 shows the datasets associated with the nodes representing $\langle A \rangle$ and $\langle A, B \rangle$ of the opening tree reported in Figure 3b. Obviously, the attribute *Completion time* is only available when training a regression model for *Completion time*, while the attribute *Next Activity* is only available when training a classification model for *Next Activity*. In the construction of the nested prediction models, any traditional machine learning algorithm for regression/classification can be used.

In the prediction phase, the algorithm traverses the (proper, contiguous or opening) frequent sequence tree in order to identify the regression/classification model to be used. The search starts from the root and proceeds towards the leaves of the sequence tree. For each new activity of a running process, the algorithm moves to the corresponding next level of the tree. If there is no corresponding

Algorithm 1. contiguous(T,n)

```
   input  : T: a sequence tree; n: node of T
   output: vil: the VIL of the sequence at node n such that the contiguous
           condition is satisfied.
 1 vil =getVIL(n); parent =getParent(n);
 2 while parent != root(T) do
 3     parentVil = getVIL(parent);
 4     foreach j = 1...length(vil);
 5     do
 6         i = 0; contiguous = FALSE;
 7         while ++i < len(vil[j]) and not contiguous do
 8             z = 0;
 9             while ++z ≤ i and not contiguous do
10                 if vil[j][i] = parentVil[j][z] + 1 then
11                     vil[j] = parentVil[j][z..(len(parentVil[j]) − 1)];
12                     contiguous = TRUE ;
13             if not contiguous then
14                 vil[j] = NULL;
15     n = parent; parent =getParent(n);
16 return vil;
```

node, that is, the complete sequence was not found frequent during the partial process model construction, the algorithm does not move to the next level and remains in the current node until a new activity of the running process allows us to move to the next level. At each point of the running process, the prediction model associated to the current node is used for predictive purposes.

Example 3. Let $p = \langle A, B, E, C\rangle$ be the running process for which we intend to predict either the next activity or the completion time. Let the sequence tree in Figure 3b be the learned partial model. Starting from the root of the tree, when the first activity of the process p arrives, the algorithm moves first in the node associated with the sequence $\langle A\rangle$, then, when the second activity of the process p arrives, it moves in the child node associated with $\langle A, B\rangle$. Since no child node of $\langle A, B\rangle$ associated with the sequence $\langle A, B, E\rangle$ exists, when the third activity of the process p arrives, the algorithm remains in the current node. When the next activity C of p arrives, the algorithm moves in the node associated with the pattern $\langle A, B, C\rangle$ and uses the nested models in this node for prediction.

3 Experiments

In this section we present the empirical evaluation of the proposed algorithm. For evaluation purposes, we used a 10-fold cross-validation schema and, we collected the average classification rate (C-RATE, i.e. the number of processes for which

Table 2. Number of sequences extracted during the sequential pattern mining phase for ProM and THINK3

minsup	40%			30%			25%		
tree type	1	2	3	1	2	3	1	2	3
Prom	607	31	6	735	53	11	4671	66	13
Think3	104	26	7	208	40	9	855	68	21

we were able to obtain predictions), the average predictive accuracy of the next activity and the error in the completion time estimation. As for this last error, we use the symmetric mean absolute percentage error (SMAPE) defined in Equation (1), since it is more robust to effect of zero or near-zero values than traditional error measures [6] and the classical root relative mean squared error (RRMSE) defined in Equation (2) [10]. In both equations y_i is the actual completion time, $\hat{y_i}$ is the estimated completion time and $\overline{y}_i$ is the average completion time.

$$SMAPE = \left(\sum_{i=1}^{n} |\hat{y_i} - y_i|\right) / \left(\sum_{i=1}^{n} (\hat{y_i} + y_i)\right) \tag{1}$$

$$RRMSE = \sqrt{\left(\sum_{i=1}^{n} (\hat{y_i} - y_i)^2\right) / \left(\sum_{i=1}^{n} (\overline{y}_i - y_i)^2\right)} \tag{2}$$

In our implementation, we use as nested learning algorithms C4.5 [8] (to predict the next activity) and M5' [15] (to estimate the completion time). Results are collected by varying the minimum support threshold and the type of the sequence tree. We denote proper sequence trees with "tree type 1", contiguous trees with "tree type 2" and opening trees with "tree type 3". Completion time results obtained with our approach are compared with results obtained from the transition systems implemented in ProM5 Suite, where the prediction is made based on the average time to completion for process instances in a similar state [13]. Unfortunately, the ProM5 Suite does not include tools for next activity prediction. Since Tree type 3 is, as stated before, the model more similar to a transition system, we consider this setting as baseline for our comparisons.

The evaluation is performed on two real datasets that is, ProM and THINK3. ProM concerns repairing telephones of a communication company. The event log contains 11,855 activities and 1,104 cases, while the number of distinct performers is 29. Activities are classified as complete (1,343), schedule (6,673), resume (178), start (809), suspend (166) and unknown (remaining). Additionally, ProM stores several properties like name, timestamp, resources (in term of roles). The second dataset, THINK3 [1] is an event log presenting 353,490 cases in a company, for a total of 1,035,119 events executed by 103 performers. Activities are classified as administrator tools (131), workflow (919,052), namemaker (106,839), delete (2,767), deleteEnt (2,354), prpDelete (471), prpSmartDelete (53), prpModify (34) and cast (1,430).

Results on ProM are extracted by using three minimum support thresholds: 0.4, 0.3 and 0.25. Table 3 shows results for both the considered predictive tasks. As it is possible to see, even if we consider a partial process model, we are

Table 3. Averaged cross-validation results for ProM with different tree type (tree type). Column "gain" indicates the RRMSE gain over the ProM5 Suite for completion time prediction.

tree type	minsup	Completion time prediction			Next activity prediction	C-RATE
		RRMSE	SMAPE	gain(%)	ACCURACY	
1	40%	0.71	0.19	29%	0.72	1.00
	30%	0.70	0.20	30%	0.74	1.00
	25%	0.69	0.19	31%	0.78	1.00
2	40%	0.83	0.24	17%	0.60	1.00
	30%	0.69	0.19	31%	0.64	1.00
	25%	0.69	0.20	31%	0.68	1.00
3	40%	0.83	0.24	17%	0.60	1.00
	30%	0.69	0.19	31%	0.64	1.00
	25%	0.69	0.20	31%	0.68	1.00

Table 4. Average running times for ProM (sec.)

tree type	minsup	seq. pattern discovery	datasets generation	construction nested model	total
1	40%	0.448	7.783	2.898	11.129
	30%	0.498	8.514	2.965	11.977
	25%	0.863	162.313	2.892	166.068
2	40%	0.448	5.478	3.155	9.081
	30%	0.498	5.892	3.707	10.097
	25%	0.863	6.666	3.434	10.963
3	40%	0.448	6.008	3.245	9.701
	30%	0.498	5.853	3.497	9.848
	25%	0.863	6.352	3.452	10.667

able to provide a prediction for (almost) all the sequences (C-RATE). Moreover, we can observe that trees of type 2 and 3 are more robust to noise and to incompleteness with respect to trees of type 3. For the next activity prediction task, by reducing minsup, predictive accuracy increases. Moreover, the partial model based on propers sequence trees (tree type 1) leads to the best results. By comparing these results with those reported in Table 2, we can see that best results are obtained with the most complete trees. This means that *minsup=0.25* is still enough to do not suffer from overfitting problems. As for completion time prediction, we show that results do not change significantly varying the tree type. Best results are obtained with proper sequence trees (i.e tree type 1) and *minsup=0.25*; contiguous trees (i.e. tree type 2) and *minsup=0.3*; opening trees (i.e. tree type 3) and *minsup=0.3*. This means that, in this case, the abstraction introduced in trees of type 2 and 3 is beneficial. Moreover, the comparison with the ProM5 Suite shows that our approach leads to reduce the error of a great margin (between 17% and 31% of the RRMSE). By analyzing results reported in Table 4, we see that the generation of trees of types 2 and 3 is significantly more efficient than that of trees of type 1. This means that, while for next activity prediction, high running times of tree type 1 are justified by effectiveness, this is not true for completion time prediction, where tree types 2 and 3 are the best solutions in terms of efficiency and effectiveness.

Table 5. Averaged cross-validation results for THINK3 with different configurations (conf.). Column "gain" indicates the RRMSE gain over the ProM5 Suite for completion time prediction.

tree type	minsup	Completion time prediction			Next activity prediction	C-RATE
		RRMSE	SMAPE	gain(%)	ACCURACY	
1	15%	0.91	0.49	9%	0.51	1.00
	10%	0.91	0.49	9%	0.54	1.00
	5%	0.97	0.73	3%	0.62	1.00
2	15%	0.94	0.41	6%	0.49	1.00
	10%	0.89	0.39	11%	0.49	1.00
	5%	0.92	0.47	8%	0.54	1.00
3	15%	0.95	0.49	5%	0.49	0.99
	10%	0.94	0.44	6%	0.49	0.99
	5%	0.94	0.41	6%	0.54	1.00

Table 6. Average running times for THINK3 (sec.)

tree type	minsup	seq. pattern discovery	datasets generation	construction nested model	total
1	15%	0.848	413.355	67.067	481.270
	10%	2.081	443.652	69.628	515.361
	5%	3.573	572.952	69.970	646.495
2	15%	0.848	343.444	74.323	418.615
	10%	2.081	351.362	72.796	426.239
	5%	3.573	369.396	74.447	447.416
3	15%	0.848	393.726	73.169	467.743
	10%	2.081	461.615	92.444	556.140
	5%	3.573	567.422	127.741	698.736

Results on THINK3 are obtained with three minimum support thresholds: 0.05, 0.1 and 0.15. In Table 5, we show results obtained for both the considered predictive tasks. Also in this case our approach is able to provide a prediction for (almost) all the sequences (C-RATE). Similarly to what observed for the ProM dataset, best results for the next activity prediction are obtained with proper sequence trees (i.e. tree type 1). As for the prediction of the completion time, the best results are obtained with the contiguous trees (i.e. tree type 2, *minsup=0.1*). This setting is also one of the best settings in terms of running times (see Table 6). Moreover, the comparison with the ProM5 Suite shows that our approach leads, as in the case of ProM data, to reduce the RRMSE in all cases (up to 11%). This is an interesting result if we consider that the THINK3 dataset has less attributes than the ProM dataset.

4 Conclusions and Future Works

This paper faces the problem of operational support in process mining and, in particular, the prediction of the next activity and of the completion time. The proposed approach is two-stepped and combines descriptive data mining for partial model mining and predictive data mining for mining nested classification/regression models. This solution provides incompleteness-robust and

non-overfitted prediction models thanks to the first phase, where a tailored sequential pattern mining algorithm is adopted. Moreover, with this method, we can apply any traditional classification/regression techniques thanks to the construction of the nested model. As can be seen, using this approach, completion time predictions can be significantly improved over state-of-the-art applications (approximately 30% with ProM Data and 11% with THINK3 Data).

For future work, we intend to extend the experiments with additional (noisy) cases, to check the effectiveness of the proposed approach to noise, and to exploit "closed" sequential pattern mining instead of frequent sequential pattern mining to further reduce the number of nested models to learn. Moreover, we intend to consider the use of other algorithms for sequential pattern mining with constraints, in addition to FAST as well as give more importance to recent activities in the model construction, as typically done in data stream mining.

Acknowledgements. This work fulfils the research objectives of the UE FP7 project MAESTRA (Grant number ICT-2013-612944). This work is also partially supported by the Italian Ministry of Economic Development (MISE) through the project LOGIN.

References

1. Appice, A., Ceci, M., Turi, A., Malerba, D.: A parallel, distributed algorithm for relational frequent pattern discovery from very large data sets. Intell. Data Anal. 15(1), 69–88 (2011)
2. Carmona, J., Cortadella, J., Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008)
3. Ceci, M., Appice, A.: Spatial associative classification: propositional vs structural approach. J. Intell. Inf. Syst. 27(3), 191–213 (2006)
4. Dongen, B., Busi, N., Pinna, G., Aalst, W.: An Iterative Algorithm for Applying the Theory of Regions in Process Mining. In: Proceedings of the Workshop on Formal Approaches to Business Processes and Web Services, pp. 36–55 (2007)
5. Folino, F., Greco, G., Guzzo, A., Pontieri, L.: Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. Data Knowl. Eng. 70(12), 1005–1029 (2011)
6. Hyndman, R.J., Koehler, A.B.: Another look at measures of forecast accuracy. International Journal of Forecasting, 679–688 (2006)
7. Medeiros, A.K., Weijters, A.J., Aalst, W.M.: Genetic process mining: An experimental evaluation. Data Min. Knowl. Discov. 14(2), 245–304 (2007)
8. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
9. Salvemini, E., Fumarola, F., Malerba, D., Han, J.: FAST sequence mining based on sparse id-lists. In: Kryszkiewicz, M., Rybinski, H., Skowron, A., Raś, Z.W. (eds.) ISMIS 2011. LNCS, vol. 6804, pp. 316–325. Springer, Heidelberg (2011)
10. Stojanova, D., Ceci, M., Appice, A., Malerba, D., Džeroski, S.: Global and local spatial autocorrelation in predictive clustering trees. In: Elomaa, T., Hollmén, J., Mannila, H. (eds.) DS 2011. LNCS, vol. 6926, pp. 307–322. Springer, Heidelberg (2011)

11. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st edn. Springer Publishing Company, Incorporated (2011)
12. van der Aalst, W.M.P., Pesic, M., Song, M.: Beyond process mining: From the past to present and future. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 38–52. Springer, Heidelberg (2010)
13. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Inf. Syst. 36(2), 450–475 (2011)
14. van der Aalst, W.M.P., Weijter, A., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16, 2004 (2003)
15. Wang, Y., Witten, I.H.: Induction of model trees for predicting continuous classes (1996)