Constructing Decision Trees from Process Logs for Performer Recommendation

Aekyung Kim, Josue Obregon, and Jae-Yoon Jung(✉)

Department of Industrial and Management Systems Engineering, Kyung Hee University, 1 Seochen-dong, Giheung-gu, Yongin, Gyeonggi, Republic of Korea
{akim1007, jobregon, jyjung}@khu.ac.kr

Abstract. This paper demonstrates that the discovery technique using historical event logs can be extended to predict business performance and recommend performers for running instances. For the prediction and recommendation, we adopt decision trees, which is a decision support tool in management science. Decision trees are commonly used to help identify the most likely alternative to reach a goal. To provide effective performer recommendation, we use several filters with previous performers and key tasks to the decision tree. These filters allow for a suitable recommendation according to the characteristics of the processes. The proposed approach is implemented on ProM framework and it is then evaluated through an experiment using real-life event logs, taken from a Dutch Financial Institute. The main contribution of this paper is to provide a real-time decision support tool by recommendation of the best performer for a target performance indicator during process execution based on historical data.

Keywords: Process mining · Performer recommendation · Decision tree

1 Introduction

Process mining is the business analytical method that allows for the analysis of business processes based on event logs [1, 2]. The basic idea is to extract knowledge from event logs recorded by information systems to improve business performance [3]. Most process mining techniques work on "post mortem" event data in that they analyze events that belong to completed cases. However, huge data are being generated in executing business processes and they need to be analyzed in (near) real-time [4, 5]. Therefore, process mining should be extended to online operational support to include real-time decision support [6].

In this paper we introduce decision trees to process mining in order to deal with the recommendation of performers. Among decision support tools, decision trees have two advantages which can be applied to performer assignment. First, decision trees simplify decision making problems by evaluating performances of competing alternatives and pruning inferior alternatives from leaf nodes. Second, decision trees have two types of nodes, chance nodes and decision nodes, which are appropriate to represent scheduled tasks in a process model and possible performers based on historical data, respectively. Furthermore, decision trees can reflect control-flow patterns such as sequence, AND-split, XOR-split and loop. We also define three filters which allow for a suitable recommendation depending on the characteristics of processes in the decision trees: *non-filter*, *k-recent performer filter* and *key-task performer filter*. Moreover, key performances of performers are predicted and the predicted performance values (e.g., the predicted remaining flow time or total labor cost according to performers) are projected onto the decision trees. The decision tree is then pruned with a running case and filter. Finally, best performers are recommended to minimize completion time or total labor cost.

The approach presented in this paper is implemented in the open-source process mining framework ProM and it is evaluated through an experiment using a real-life event log, taken from a Dutch Financial Institute. The proposed approach is also expected to be applied to manage business processes or projects in which performer assignment has a critical effect on business performance (e.g., software development project, troubleshooting process, and product repair process) [7]. The recommendation technique for performers presented in this paper can also be easily extended for business roles (e.g. skill levels of software engineers) and organizations (e.g. competitive teams).

2 Related Work

Process mining is a process management technique that allows for the analysis of business processes based on logs. van der Aalst categorized three types of process mining such as discovery, conformance and enhancement [3]. The last type is also regarded as operational support. He also suggests three ways of process mining to provide operational support: detect, predict and recommend. For the second type of operational support, van der Aalst et al. demonstrated that the discovered process models could be extended with information to predict the completion time of running instances [8]. For the third type of operational support, Schonenberg et al. proposed a recommendation service that could be used in combination with flexible process-aware information systems to support end users during process execution by giving recommendations on possible next steps [9].

There is also related work on job dispatching and task assigning to support decision making in business process management. Ha et al. devised process execution rules using individual worklists and proposed task assignment algorithms to maximize the overall process efficiency under the limitation of the agent's capacity [10].

The proposed approach in this paper differs from existing approaches. First of all, we predict performance indicators based on historical data. Second, we introduce a graph-based decision support tool (i.e. decision tree) to predict key performances by analyzing historical data. Finally, compared to existing process mining algorithms, the proposed approach can be applied to real-time decision support (i.e. filtered decision tree) to recommend the best performer to optimize the chosen performance indicator such as cost and time based on historical data.

3 Performer Recommendation Using Process Mining

This section describes the overall procedure of the proposed technique as shown in Fig. 1. There are information systems supporting and executing real operational processes. Events which occur in operational business processes are used to discover a process model. For process discovery, many process discovery algorithms have been proposed so far by many researchers. In our approach a decision tree is constructed from the given event log. Based on the historical data, performances are then predicted and annotated with information about times and costs (e.g., the average time and cost spent in a particular task) on the decision tree. Subsequently, the decision tree is matched with a running case and filtered according to process characteristics. Finally, the decision tree allows us to evaluate performances and recommend the best performer for the next possible tasks in terms of time and cost (e.g., the best performer in terms of completion time, or the best performer in terms of total cost).

4 Decision Tree Construction for Performance Prediction

In this section, we describe how to construct a decision tree for performance prediction based on an event log. To apply decision trees to the problem, tasks and performers are considered as environmental factors and decision factors, respectively. Therefore, in decision trees, tasks are represented by decision nodes which are depicted by squares while performers are represented by chance nodes which are depicted by circles. Then the information of predicted performances such as processing time and labor cost is annotated on the decision tree.

To explain how to predict performances, we first define the concept of a performer-oriented event log, a performer-oriented activity log and their annotation. Then we explain how to construct a decision tree from historical process data.

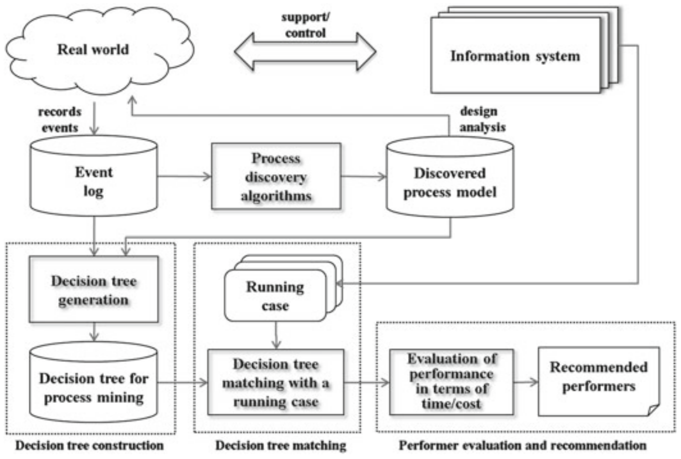

Real world
support/ control
Information system
records events
design analysis
Event log
Process discovery algorithms
Discovered process model
Decision tree generation
Running case
Decision tree for process mining
Decision tree matching with a running case
Evaluation of performance in terms of time/cost
Recommended performers
Decision tree construction
Decision tree matching
Performer evaluation and recommendation

Fig. 1. Overview of performance recommendation based on historical data.

4.1 Historical Process Model

We first define the concept of a historical process data. To be able to reason about event logs and precisely specify the requirements for the logs, many researchers proposed their formal models of process mining. In this research, we adopt and extend the formal models presented by van der Aalst [3]. The performer-oriented event log in Definition 1 is described to define the performer-oriented event and case [11].

Definition 1 (Performer-oriented event log). Let $\mathscr{C}$ and $\mathscr{N}$ be the sets of all possible case identifiers and activity names, respectively. For any case $c \in \mathscr{C}$ and name $n \in \mathscr{N}$, $\#_n(c)$ is the value of attribute n for case c. Each case has a special mandatory attribute trace: $\#_{\mathrm{trace}}(\mathrm{c}) \in \mathscr{E}^*$ where $\mathscr{E}^*$ is the set of all finite sequences over the set of all possible event identifiers. $\hat{c} = \#_{\mathrm{trace}}(c)$ is a shorthand for referring to the trace of a case. A *performer-oriented event* $\underline{e}$ is an event which is characterized by a performer. A *performer-oriented trace* of case $\underline{\hat{c}}$ is a trace of a *performer-oriented event* $\underline{e}$, and a *performer-oriented event log* $\underline{L} \subseteq \mathscr{C}$ is a bag of performer-oriented cases.

Performer-oriented events are identified by their activity names, performers, time and transactions, i.e., *performer-oriented event* $\underline{e} = (\#_{\mathrm{activity}}(e), \#_{\mathrm{performer}}(e), \#_{\mathrm{time}}(e), \#_{\mathrm{trans}}(e))$. Also, a *performer-oriented case* $\underline{\hat{c}} = <\underline{e_1, e_2, \ldots e_n}>$ and a *performer-oriented event log* $\underline{L} = [\underline{\hat{c}}]$. In this paper, we apply two following operators for considering processing time and labor cost: Given an activity identifier a = #activity(e), *operator1* is "#processingtime(a) = #completetime(a) − #starttime(a)" and *operator2* is "#labor_cost(a) = #performer(a).unitcost × #processingime(a)". Here #performer(a).unitcost means labor cost per an hour of #performer(a).

Definition 2 (Performer-oriented activity log). Let $\underline{L} \subseteq \mathscr{C}$ be a bag of *performer-oriented cases* as defined in Definition 1. If two operators are applied to a *performer-oriented event* $\underline{e}$, then a *performer-oriented activity model* (annotated with processing time and labor cost) $\underline{a} = (\#_{\mathrm{name}}(a), \#_{\mathrm{performer}}(a), \#_{\mathrm{processingtime}}(a), \#_{\mathrm{laborcost}}(a))$ and a *performer-oriented activity trace* $\hat{m}_c = <\underline{a_1}, \underline{a_2}, \underline{a_3}, \ldots>$ i.e., sequence of $\underline{a}$, are obtained. Therefore a *performer-oriented activity log* $\underline{M} = [\hat{m}_c]$, i.e., a bag of *performer-oriented activity traces* $\hat{m}_c$, is defined.

4.2 Construction of Decision Tree

In this section we explain the procedure of creating a decision tree for performance prediction based on historical data. First we define formally the concept of performer-oriented decision tree.

Definition 3 (Performer-oriented decision tree). A *performer-oriented decision tree* is a tuple pDt = (T, t_0, P, F, E) where T is the set of *task nodes*, $t_0 \in$ T is the *root node* of the *decision tree*, P is the set of *performer nodes*, F is the set of *final nodes* and E $\subseteq$ (T × P) ∪ (P × T) ∪ (P × F) is the set of directed arcs that determines the flow direction of the tree. We assume that nodes are characterized for some attributes, e.g., *task nodes* have a label attribute and *performer nodes* have a label attribute and a processing time attribute. For any $t \in T$ and $p \in P$, $\#_n(t)$ or $\#_n(p)$ represent the value of attribute n for the node t or p. For example $\#_{label}(node)$ refers to the value of the attribute label of *node.*

Now let us define the procedure for constructing a *performer-oriented decision tree* pDt based on historical process data. Having a *performer-oriented activity log* $\underline{M}$ we define the steps for creating the decision tree as follows:

1. Create an empty *performer-oriented decision tree* pDt and a node called *current node* that will point to the current *task node* traversed on pDt.
2. Start traversing every $\hat{m}_c \in \underline{M}$ and every $\underline{a} \in \hat{m}_c$.
3. If t_0 is empty, it means that is the first activity read from the activity log therefore a new node is created and added as root of the decision tree i.e., t_0 of pDt. The creation of new nodes is as follows:
   a. Create a new task node *tnode* assigning $\#_{\text{label}}(tnode) = \#_{\text{name}}(a)$.
   b. Create a new performer node *pnode* assigning $\#_{\text{label}}(pnode) = \#_{\text{performer}}(a)$ and $\#_{\text{processingtime}}(pnode) = \#_{\text{processingtime}}(a)$.
   c. Assign *pnode* as child of *tnode*.
   d. Assign *tnode* as *current node*.
4. If t_0 is not empty, it means the tree is being traversed.

   a. Define a task node n that can take two possible values: the value of *current node* or the value of any task node connected to *current node* by means of its performer child node such that $\#_{\text{label}}(current\ node\text{-}child) = \#_{\text{performer}}(a)$. After that, three possible cases could occur

      i. Node n such that $\#_{\text{label}}(n) = \#_{\text{name}}(a)$ does not exist. In this case the steps for new node creation are repeated (i.e., 3a, 3b, 3c and 3d) and the new node is added to pDt.
      ii. Node n such that $\#_{\text{label}}(n) = \#_{\text{name}}(a)$ already exists, but the node n does not have a child *nchild* that satisfy $\#_{\text{label}}(nchild) = \#_{\text{performer}}(a)$. In this case a new performer node is created assigning $\#_{\text{label}}(pnode) = \#_{\text{performer}}(a)$ and $\#_{\text{processingtime}}\ (pnode) = \#_{\text{processingtime}}(a)$. Then the new performer node is added as a child of the node n.
      iii. Node n such that $\#_{\text{label}}(n) = \#_{\text{name}}(a)$ already exists, and node n has a child *nchild* that satisfy $\#_{\text{label}}(nchild) = \#_{\text{performer}}(a)$. In this case add a new processing time to the bag of processing times $\mathbb{B}(pt)$ of *nchild* such as $\#_{\text{processingtime}}(nchild) = \#_{\text{processingtime}}(a)$.

After completing the steps given before, we have each performer node on the decision tree annotated with a bag of processing times $\mathbb{B}(pt)$. Thus each processing time $pt \in \mathbb{B}(pt)$ represents a specific case in which the performer appears into the *performer-oriented activity log* $\underline{M}$.

The next step is to calculate expected values of KPI, such as predicted processing time of individual performers, represented as *performer nodes* on the constructed *performer-oriented decision tree*. In this paper we combine the approach used in [8] with our approach as follows. First we have to convert the bag of processing times $\mathbb{B}(pt)$ into a single value. For this we use the same prediction function used in [8], in which the authors state that the bag of a chance node should be seen as a sample of

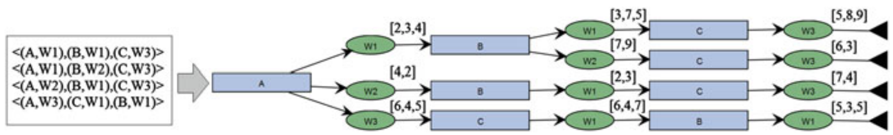


Fig. 2. A simple annotated decision tree with each chance node, representing a performer, annotated with processing times.

times. Based on this, the sample mean (i.e., $\overline{pt} = \sum_{i=1}^{n} pt_i/n$) is a good estimator of the population mean of processing times. Moreover, predicted processing times could be calculated independent or dependent on previous performers.

Let us define first how to calculate predicted processing time independently. If processing time is assumed to be independent on previous performers, the predicted processing time of the pending task by a particular performer is given by the average of the processing times contained in the union of bags $\mathbb{B}(pt)$ belonging to all the cases corresponding to the same performer, that have the same previous partial trace σ without taking in consideration the performer who executed each previous activity in σ. For example, in Fig. 2, if we want to calculate predicted processing time assuming independent of previous performers of *W1* in task *B* in the first path of the decision tree, we have to take in consideration traces σ_1 <(A,W1) (B,W1) (C,W3)> and σ_3 <(A,W2) (B,W1) (C,W3)>. The bag of the first trace is $\mathbb{B}(\sigma_1) = [3, 5, 7]$ and the bag of the second trace is $\mathbb{B}(\sigma_3) = [2, 3]$. As we defined before, the predicted processing time of *W1* on task *B*, assuming independent of previous performers, is calculated with the average of the union of bags of σ_1 and σ_3 ($\mathbb{B}(\sigma_1) \cup \mathbb{B}(\sigma_3) = [3, 7, 5, 2, 3]$). Therefore the predicted processing time of *W1* in task *B* is 4 units of time. It is important to remark that the last trace of Fig. 2, σ_4 <(A,W1) (C,W1) (B,W1)> is not taken in consideration in the calculation of predicted processing time assuming independency of previous performers, although σ_4 has a performer *W1* executing task *B*, the previous partial trace is not the same than σ_1 and σ_3.

Now we define how to calculate predicted processing time dependently. If processing time is assumed to be dependent on previous performers, the predicted processing time of the pending task by a particular performer is given by the average of the annotated processing times bag $\mathbb{B}(pt)$, belonging to the cases corresponding to the same performer, that have exactly the same previous partial trace σ, i.e. the task and the performer who executed each previous activity in σ should be the same.

5 Decision Tree Matching with Running Case

5.1 Running Process Case

We first present a model for running cases that can be monitored in process-aware information systems. During process execution, a partial trace can be captured through a running process. To represent the partial trace, the information of the running case is defined as a partial trace of a *performer-oriented activity trace* of case.

Definition 4 (Running case). A running case is represented as a partial trace of *performer-oriented activity trace* of a case. A partial trace of a running case σ_{p} produces some representation. Formally, $\sigma_{\mathrm{p}} \subseteq \hat{m}_c$ where $\hat{m}_c$ is a *performer-oriented activity trace*. A pending task, i.e., candidates of next task, is represented as $t \in A$ where A is a set of activities.

5.2 Decision Tree Matching

We now explain how to match a constructed decision tree with a running case. The procedure of matching a decision tree with running cases is composed of three steps.

The first step is to select a filter and extract the subtrees from pDt according to the characteristics of the filter selected. Business processes include many tasks that do not affect the performance of performers because the tasks are not related between them. For this reason, we propose three kinds of filters that help to enhance the effectiveness of performance prediction. The filtering seeks to include only the tasks that have a considerable effect on predicting the value of KPI.

We propose three kinds of filters: *non-filter, k-recent performer filter* and *key-task performer filter*. The *non-filter* does not apply any special filtering over a decision tree and it considers all cases that have exactly the same partial trace σ than the running case. If the *non-filter* is selected, only a single subtree is extracted from the constructed decision tree. The *k-recent performer filter* filters a decision tree with k-recent performers. The filter considers that k-recent performers of a task affect the performance of the performer for a pending task. If the *k-recent performer filter* is selected, one or more subtrees are extracted from the constructed decision tree. The *key-task performer filter* filters a decision tree with the performers of the key tasks. It considers that the performers of important tasks affect the performance of the performer on a pending task. The extracted subtrees are merged into a single subtree. If one assumes independence on previous performers, the values of predicted processing time do not need to be updated, whereas if one assumes dependence on previous performers, the values of predicted processing time need to be recalculated and updated combining the bags of processing times $\mathbb{B}(pt)$ of the nodes merged corresponding to the same task and/or same performer depending on the filter selected.

Finally, we can evaluate and recommend performers using the performer-oriented decision tree. The procedure for recommendation of the best performer for a given a running case has two steps:

1. *Evaluate performer,* in this step performers are evaluated based on the predicted processing times calculated in previous steps. The filtered performer-oriented decision tree is traversed backwards. Starting from *final nodes*, a new attribute called *total remaining time* is calculated and added to the node being traversed. The attribute *total remaining time* is equal to the summation of the current node traversed *processing time* attribute value plus the minimum *total remaining time* attribute among its children nodes. From here it depends on which value of KPI we want to apply. If the KPI value concerns only on remaining time, we use the attribute *total remaining time* for evaluation. However, if the KPI value concerns about cost, the evaluation is based on the attribute *labor cost* defined in previous sections.

2. *Recommend the best performer*, the performer of the pending task who has the highest predicted value of performance indicator in the evaluation step is then recommended.

6 Implementation and Experiment

The technique presented in this paper was implemented as a plug-in for the ProM Framework. The ProM framework integrates the functionality of several existing process mining tools and provides additional process mining plug-ins [12, 13].

In this paper we implement the plug-in called *DTMiner* which takes an event log as an input. The plug-in constructs a *performer-oriented decision tree* based on an event log using the algorithm defined in previous sections. As an initial result, a *performer-oriented decision tree* is displayed on the screen as shown in Fig. 5. *Task nodes* are filled with blue color and *performer nodes* with green color. Node information such as processing time is displayed when the mouse pointer hovers on the node. The edge connection between task and performer nodes displays the predicted processing time taken by the performer to reach the next task.

The analysis section of the plug-in is displayed at the bottom of the screen as shown in Fig. 5. The completed activities of target running cases can be edited (i.e., added or removed) from here. Moreover different filter actions can be executed over the decision tree constructed as described in previous sections (e.g., *non-filter filter* or *k-recent performer filter*). Finally recommendation is provided to the user showing the recommended paths by means of filling nodes and coloring edges with color red as shown in Fig. 5. The predicted total remaining time (i.e. the predicted remaining time needed to finish the case) for each decision and chance node is calculated and showed when the mouse pointer hovers on the nodes.

We demonstrate the applicability of our approach using a real-life event log using *DTMiner* plug-in. For the experiment we used the real-life event log of a Dutch Financial Institute which was prepared by Business Process Intelligence Challenge (BPIC'12). We first explain the log and describe how to preprocess the log for experiment. Our approach is then validated and tested using the log.

6.1 Process Discovery from Event Log

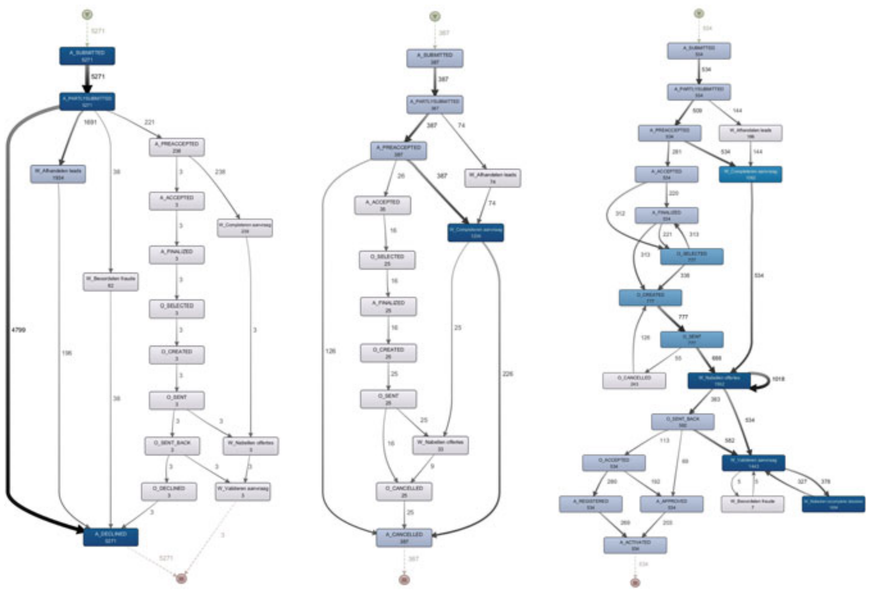
As mentioned on the website[1] of BPIC, this log contains 13,087 cases and 262,200 events over a six month period from October 2011 to March 2012. The process represented in the event log is an application process for a personal loan or overdraft within a global financing organization.

To provide some insight into the structure of the process, we have used Disco which draws process models using the improved fuzzy algorithm. It also shows

[1] See http://www.win.tue.nl/bpi2012/doku.php?id=challenge

meaningful information such as variants, frequency, and duration and provides powerful filtering features. We found that the whole process could be split into three sub-processes by end events (i.e. *A_DECLINED, A_CANCELLED, A_ACTIVATED*). In the next subsection, we experiment using the three groups split from whole cases. The group that ends with *A_DECLINED* has 7,635 cases. The *A_CANCELLED* group has 2,807 cases and the *A_ACTIVATED* group has 2,246 cases.

Before testing our approach the log needs to be preprocessed, e.g., event and cases may be removed for various reasons. There are some events in the log where the resource information is missing. We first removed all the cases that have at least one event with NULL resource information because they cannot be used for the performer recommendation method. Also, we removed all cases which have at least one event with performers who have lower than 5 % of relative frequency. Second, we used only cases whose sequence of activities is shared by at least 10 cases using variation filtering functionality of Disco. Moreover, we consolidated all the resources performed in automatic activities which have zero duration into a resource called 'Automatic'. Finally, we split the filtered log into three sub-processes. As a result the group that ends with *A_DECLINED* has 5280 cases. The *A_CANCELLED* group has 1024 cases and *A_ACTIVATED* group has 534 cases after filtering. Figure 3 shows the process models of each group discovered by Disco.



(a) *A_DECLINED* (b) *A_CANCELLED* (c) *A_ACTIVATED*

Fig. 3. The sub-process models discovered from Dutch Financial Institute's log

6.2 Performance Recommendation

In this subsection, we present an example scenario with the log of Dutch Financial Institute to describe how the proposed approach can be applied to performer's allocation problem with our plug-in for ProM. Using the plug-in with the example scenario, the historical process log of a business process was analyzed to construct the decision tree, which was used to recommend the best performers for an ongoing instance of the process.

To improve the quality of the customer loan service, the Dutch Financial Institute would want to reduce the lead time of their services. At the same time, they would want to decrease the cost of their processes. For this reason, the purpose of this experiment is to recommend the best performer who allows the remaining time and the total labor cost to be reduced for each possible next task.

As depicted in Fig. 1 the overall procedure of the proposed approach consists of three primary steps. Following these steps, we first constructed decision trees from the log using our *DTMiner* plug-in. In this step, we used three sub-logs and constructed decision trees separately. Figure 4 shows the decision trees constructed from the sub-process that ends with '*A_DECLINED*' and the annotated time is calculated by option of 'dependence'.

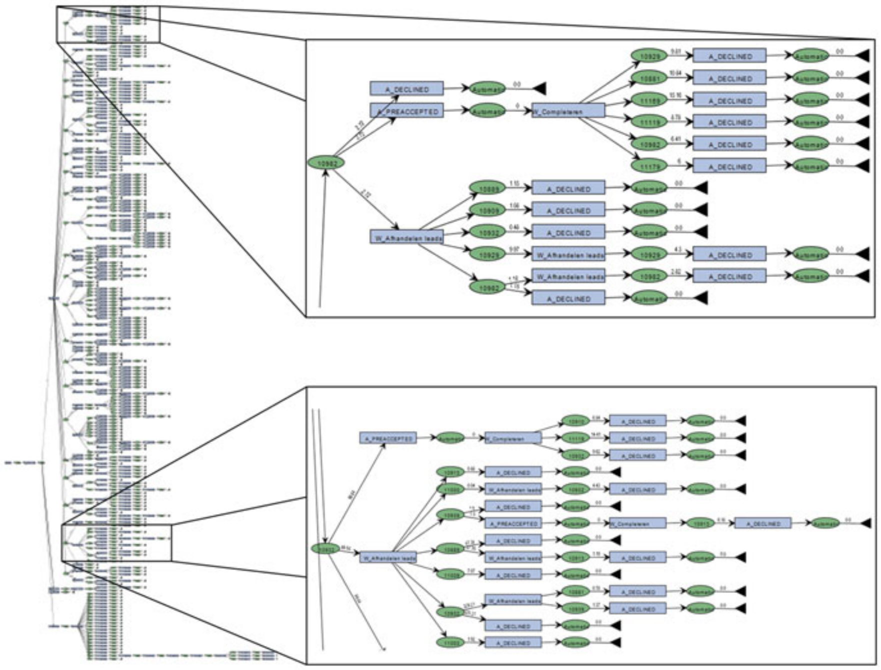
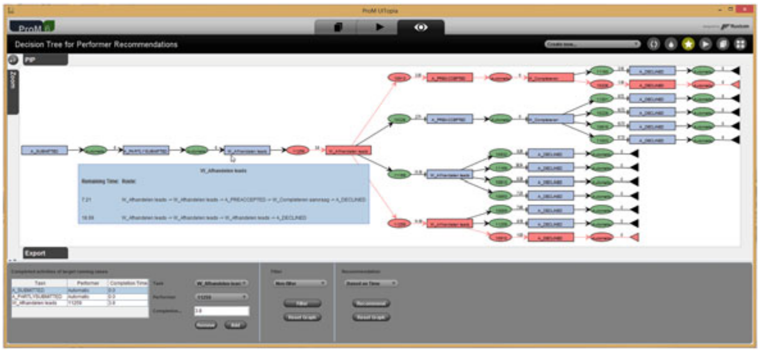


Fig. 4. The constructed decision tree from the sub-process that ends with '*A_DECLINED*'

In the second step, we considered that one running case σ_p = (*A_SUBMITTED*, Automatic, 0, 0) (*A_PARTLYSUBMITTED*, Automatic, 0, 0) (*W_Afhandelen lead*, 11259, 3.8, 8)> was captured by information system. Also, we assumed that a manager did not want to filter with previous performers, and he wanted to obtain the recommended performers who can reduce the remaining time. Then, we set up the running case and filter options as shown in the bottom of Fig. 5.

In the last step, information about the running case was matched with the decision trees and its subtrees were extracted and merged. Also, the predicted KPIs were updated. Finally, we evaluated performance and recommended the best performer for each next possible task by means of reducing inferior performers from leaf nodes. Figure 5 shows the pruned decision tree and the best performer of each task in sub-process that ends with '*A_DECLINED*'.

After executing the running case, the pruned decision tree showed two possible traces with different excution probabilities as shown in Fig. 5. The first trace σ_1 = <*A_SUBMITTED*, *A_PARTLYSUBMITTED*, *W_Afhandelen lead*, *W_Afhandelen lead*, *A_PREACCEPTED*, *W_Completeren aanvraag*, *A_DECLINED*> had an execution probability of 40 % and the second trace σ_2 = <*A_SUBMITTED*, *A_PARTLYSUBMITTED*, *W_Afhandelen lead*, *W_Afhandelen lead*, *W_Afhandelen lead, A_DECLINED*> had an execution probability of 60 %. Based on this probabilities, we can recommend the best performer for task '*W_Afhandelen lead*' is '10913', and the best performer for task '*W_Completeren aanvraag'* is '10228' in σ_1 in which the remaining time is 3.41 and is also the minimum remaining time. In the same way, the best performer for task '*W_Afhandelen lead*' is '11259' in σ_2. Also, Fig. 6 shows performer evaluation and recommendation for the sub-process that ends with'*A_CANCLLED*' when a running case σ_p = (*A_SUBMITTED*, Automatic, 0, 0) (*A_PARTLYSUBMITTED*, Automatic, 0, 0) (*W_Afhandelen lead*, 10982, 2.3, 9) (*A_PREACCEPTED*, Automatic, 0, 0)> is given and *3-recent filter* is selected.



W_Afhandelen leads

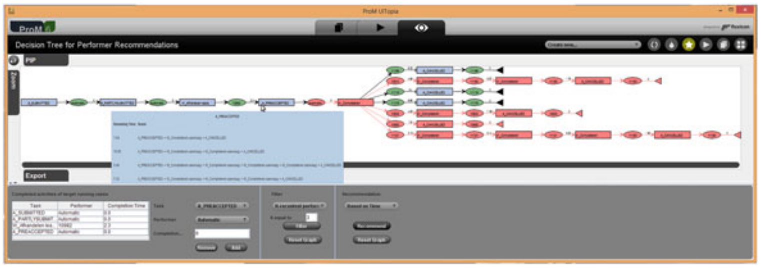
Remaining Time	Route
7.21	W_Afhandelen leads -> W_Afhandelen leads -> A_PREACCEPTED -> W_Completeren aanvraag -> A_DECLINED
18.08	W_Afhandelen leads -> W_Afhandelen leads -> W_Afhandelen leads -> A_DECLINED

Completed activities of target running cases

Task	Performer	Completion Time
A_SUBMITTED	Automatic	0.0
A_PARTLYSUBMITTED	Automatic	0.0
W_Afhandelen leads	11259	3.8

Task: W_Afhandelen leads
Performer: 11259
Completion...: 3.8
Filter: Non-filter
Recommendation: Based on Time

Fig. 5. Performer evaluation and recommendation for the sub-process that ends with '*A_DECLINED*'



Task	Performer	Completion Time
A_SUBMITTED	Automatic	0.0
A_PARTLYSUBMIT	Automatic	0.0
W_Afhandelen lea...	10982	2.3
A_PREACCEPTED	Automatic	0.0

Fig. 6. Performer evaluation and recommendation for the sub-process that ends with '*A_CANCELLED*'

7 Conclusion

In this paper, we presented a new approach that constructs decision trees based on process event log for performer recommendation. Given a running case, our approach to performer recommendation allows answering questions such as "Who is the best performer to minimize remaining time until completion?" and "Who is the best performer to save additional labor costs?" Those questions are dependent on various aspects, e.g., orders or frequencies of performers. The solutions to the best performers are discovered by considering the predicted values of KPI based on historical data. To solve the problem, decision trees are constructed based on historical process events to support simple decision making for performance recommendation.

One of the main contributions of this paper lies in a real-time decision support technique for ongoing processes. Although existing decision support methods evaluate alternatives in run-time, the proposed technique can recommend an alternative of performers for every task in a running case during process execution based on the probabilistic data of historical events.

Acknowledgments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (Nos. 2012R1A1B4003505 and 2013R1A2A2A03014718).

References

1. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., de Medeiros, A.K.A., Song, M., Verbeek, H.M.W.: Business process mining: an industrial application. Inform. Syst. **32**(5), 713–732 (2007)
2. de Medeiros, A.K.A., Pedrinaci, C., van der Aalst, W.M., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An outlook on semantic business process mining and monitoring. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2007, Part II. LNCS, vol. 4806, pp. 1244–1255. Springer, Heidelberg (2007)
3. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011)

4. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.-C.: Business process intelligence. Comput. Ind. **53**(3), 321–343 (2004)
5. Kang, B., Cho, N.W., Kang, S.-H.: Real-time risk measurement for business activity monitoring (BAM). Int. J. Innov. Comp. I. **5**(11A), 3647–3657 (2009)
6. Buytendijk, F., Flint, D.: How BAM can turn a business into a real-time enterprise. Gartner Research, AV-15-4650 (2002)
7. Bose, R.: Understanding management data systems for enterprise performance management. Ind. Manage. Data Syst. **106**(1), 43–59 (2006)
8. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Inform. Syst. **36**(2), 450–475 (2011)
9. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.: Supporting flexible processes through recommendations based on history. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
10. Ha, B.-H., Bae, J., Park, Y.T., Kang, S.-H.: Development of process execution rules for workload balancing on agents. Data Knowl. Eng. **56**(1), 64–84 (2006)
11. Kim, A., Jung, J.-Y.: A process mining technique for performer recommendation using decision tree. In: Korean Institute of Industrial Engineers Conference (2012)
12. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H., Weijters, A., van der Aalst, W.M.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
13. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: ProM 6: the process mining toolkit. BPM Demonstration Track **615**, 34–39 (2010)