Process remaining time prediction using query catalogs

Alfredo Bolt[1] and Marcos Sepúlveda[2]

(1) School of Engineering
Universidad Finis Terrae
Pedro de Valdivia 1509, Santiago, Chile
abolti@uft.edu

(2) Computer Science Department
School of Engineering
Pontificia Universidad Católica de Chile
Vicuña Mackenna 4860, Santiago, Chile
marcos@ing.puc.cl

Abstract. A relevant topic in business process management is the ability to predict the outcome of a process in order to establish *a priori* recommendations about how to go forward from a certain point in the process. Recommendations are made based on different predictions, like the process remaining time or the process cost. Predicting remaining time is an issue that has been addressed by few authors, whose approaches have limitations inherent to their designs. This article presents a novel approach for predicting process remaining time based on query catalogs that store the information of process events in the form of partial trace tails, which are then used to estimate the remaining time of new executions of the process, ensuring greater accuracy, flexibility and dynamism that the best methods currently available. This was tested in both simulated and real process event logs. The methods defined in this article may be incorporated into recommendation systems to give a better estimation of process remaining time, allowing them to dynamically learn with each new trace passing through the system.

Keywords: Process mining, process remaining time prediction, query catalogs.

1 Introduction

When dealing with service processes, a commonly asked question is: when will this case finish? Most companies give their customers general answers, such as "this process usually takes between 3 or 4 weeks". Unfortunately, in many cases this answer is not sufficient for the customers, as it is just an average cycle time plus an expected variation. It is simple, but it lacks precision and it does not consider the activities that have already been executed to give a more accurate prediction. In this article, we present a novel approach to make remaining time predictions, with a better precision than previous approaches and the ability to adjust the remaining time pre-

© Springer-Verlag Berlin Heidelberg 2011

dictions based on new information about the process execution that is being stored by the information systems that support the process.

During the last decades, information systems have been enabling organizations to improve and change their processes. Today, process-aware information systems (PAIS) [1] are able to store data of daily process operations. This information is stored as process events in an event log. When an organization has a PAIS, in most cases the information stored in these event logs is never used. In some cases, supported by specific vendors, the information is used to obtain general process information, such as some business metrics. In a few cases, non-trivial information is extracted using process mining techniques. The extracted information sometimes can be very useful for analyzing and understanding the structure and behavior of a process.

Process mining covers a wide range of techniques, from process discovery to making predictions and recommendations, many of them with a lot of applications [2]. To build recommendation services such as [3], certain predictions must be made to suggest the way forward.

The remaining time of a process instance is an important prediction for any recommendation system. Another valuable use of this prediction is to relate it to a business rule, in order to provide important information for decision making or to modify a process behavior. The approach presented in this article focuses on making process remaining time predictions using query catalogs, overcoming the inherent limitations of other approaches, such as the annotated transition system used in [4] and the regression model used in [5], providing accurate estimates using a flexible and dynamic structure.

2 Related work

Several works have addressed the task of predicting remaining time, however most were not designed to deal with the complexities of real business processes, but were oriented to more limited contexts and focuses, For example [6] focuses in on predicting cycle times in a semiconductor factory using data mining techniques. However, there are two generic approaches that were designed to predict the remaining time of any process instances: one using annotated transition systems [4] and the other using non-linear regressions [5]. Both methods outperformed simple heuristics (like the difference between the elapsed time of a process instance and its average cycle time) on the accuracy of their results, but both have limitations inherent to their design in terms of its flexibility, dynamism and ability to analyze unstructured processes.

The advantage of using annotated transitions systems [4] is that they simplify the calculation logic, and they achieve a good accuracy. The disadvantage is that the optimal selection of parameters (comparison types and horizon) cannot be known a priori, so it is bound to constant trial and error. In addition, to incorporate new knowledge into the system, e.g., a trace with an event that has never happened before.

The advantage of using non-linear regression models [5] is that it is abstracted from a process structure, and use historical data to predict the remaining time with relatively good accuracy. This allows introducing the concepts of flexibility and dy-

namism in the calculation models, which had not been addressed yet. However, the disadvantage of this approach is the same as [4] mentioned above.

3 Proposal

This article presents an approach that incorporates useful elements from annotated transitions systems [4] and creates a collection of annotated "partial trace tails" (described in 4.4) named Query Catalogs.

For this purpose, query catalogs will be created considering all possible combinations of events for all traces of an event log. This schema allows reducing the amount of processing capacity used when calculating remaining time, but it requires more memory usage.

4 Definitions

4.1 Event

An event is an action stored in a log, for example, start, completion or cancelation of an activity for a particular process instance. An event can be described by various attributes, but for the purposes of this article, we need only three, which must be present to properly use the methods proposed.

Let E be an event log and e an event. We define $\mathrm{trace\_id(e)}$ as the identifier of the trace associated with the event e, $\mathrm{act\_name(e)}$ the name of the activity associated with the event e, $\mathrm{timestamp(e)}$ the timestamp of event e, and $\mathrm{remtime(e)}$ the remaining time after the event e. Then, notice that an event log E is a sequence of events sorted in ascending order by their $\mathrm{timestamp}$ attribute.

4.2 Trace

A trace can be described through a sequence of n events T_n, with $n > 0$ which contains all the n events that have the same $\mathrm{trace\_id(e)}$, sorted in ascending order by their $\mathrm{timestamp(e)}$ attribute. Notice that there cannot be two or more traces with the same $\mathrm{trace\_id(e)}$.

For all the events e of the trace T, the remaining time $\mathrm{remtime(e)}$ can be calculated as: $\mathrm{remtime}(e_i) = \mathrm{timestamp}(e_n) - \mathrm{timestamp}(e_i)$.

4.3 Partial Trace

A partial trace PT_m is a sequence that contains the first m elements of a trace T_n, where m is the length of the partial trace and $m \leq n$ and $m > 0$.

4.4 Partial trace tail

A partial trace tail PTT_h is a sequence that contains the last h elements of a partial trace PT_m where h is the partial trace tail **horizon**, and $h \leq m$ and $h > 0$.

Then, the remaining time of a partial trace tail, $\text{remtime}(PTT_h)$, is defined as the remaining time after its last event.

4.5 Comparison types

For purposes of calculating the remaining time of a new partial trace tail, it's necessary to compare it to the partial trace tails stored in the query catalogs, then identify the similar ones to make an estimation.

First, the equivalence of two events must be defined. Let e_1 and e_2 be two events. e_1 and e_2 are defined as equivalent if they correspond to the same activity: $\text{act\_name}(e_1) = \text{act\_name}(e_2)$.

Now, to define the equivalence between partial trace tails, the comparison type that will be used to establish the equivalence must be specified:

Sequence:

Two partial trace tails are equivalent as a sequence if they have the same ordered sequence of equivalent events, e.g. a partial trace tail with 3 events $\langle A, B, C \rangle$ is only equivalent as a sequence to another partial trace tail with the events $\langle A, B, C \rangle$.

Then, the function **SequenceEquivalent**(PTT, PTT') is defined to verify if two partial trace tails are equivalent as a sequence.

Multiset:

Two partial trace tails are equivalent as a multiset, if they have the same amount of equivalent events, regardless of their sequence. e.g. a partial trace tail with 3 events $\langle A, B, C \rangle$ is equivalent as a multiset to the following partial trace tails: $\langle A, B, C \rangle$, $\langle A, C, B \rangle$, $\langle B, C, A \rangle$, $\langle B, A, C \rangle$, $\langle C, B, A \rangle$, $\langle C, A, B \rangle$

Then, the function **MultisetEquivalent**(PTT, PTT') is defined to verify if two partial trace tails are equivalent as a multiset.

Set:

Two partial trace tails are equivalent as a set if each event in a partial trace tail has at least one equivalent event in the other partial trace tail and vice versa, regardless of their sequence and how many times the events had occured. e.g. a partial trace tail with 3 events $\langle A, B, C \rangle$ is equivalent as a multiset to the following partial trace tails: $\langle A, B, C \rangle$, $\langle A, C, B \rangle$, $\langle B, C, A, B \rangle$, $\langle C, C, A, B \rangle$, etc..

Then, the function **SetEquivalent**(PTT, PTT') is defined to verify if two partial trace tails are equivalent as a set.

5 Description of the query catalog approach

With all previous definitions made, we can now describe the structure and algorithms that define the query catalog approach, and how it is used to predict remaining times.

5.1 Query catalog structure

A query catalog Q is a group of non-equivalent partial trace tails from all traces that have occurred in an event log, and additional information about each partial trace tail. In order to simplify search operations, three catalogs are used, one for each comparison type. Along with every partial trace tail $\mathrm{PTT_i}$ stored in each catalog, the number of times it has occurred ($\mathrm{q_i}$) and the sum of its remaining times ($\mathrm{s_i}$) are also stored in each catalog. It is then straightforward to calculate the average remaining time for each partial trace tail. So, the structure of every query catalog containing n partial trace tails will be: $\mathrm{Q_{sequence/multiset/set}} = \{(\mathrm{PTT}^1, \mathrm{q}_1, \mathrm{s}_1), \cdots, (\mathrm{PTT}^n, \mathrm{q_n}, \mathrm{s_n})\}$

Notice that the same partial trace tail can exist in more than one catalog, but it cannot exist more than once inside the same catalog.

5.2 Creating/Updating query catalogs

This article considers the existence of 3 query catalogs: $\mathrm{Q_{sequence}}$, $\mathrm{Q_{multiset}}$ and $\mathrm{Q_{set}}$. In order to simplify the following definitions, Q is definied as the generalization of the 3 query catalogs mentioned above.

Function **ExistsPartialTraceTail**(PTT, Q) is defined to verify if the partial trace tail being has an equivalent partial trace tail in the query catalog by cheking every partial trace tail in the query catalog. This algorithm returns the position of the PTT' found in the query catalog Q.

Function **AddPartialTraceTail(**PTT, Q**)** is defined to add a new PTT to the query catalog Q. The new element added is $(\mathrm{PTT}, 1, remtime(\mathrm{PTT}))$.

Function **UpdatePartialTraceTail(**PTT, Q, i**)** is defined to update the information ($\mathrm{q_i}$ and $\mathrm{s_i}$) of existing $\mathrm{PPT}^{\boldsymbol{i}}$ in the query catalog Q with the information of the new PPT. This algorithm updates the element $(\mathrm{PPT}^{\boldsymbol{i}}, \mathrm{q_i}, \mathrm{s_i})$ in the catalog as follows: $q_i = q_i + 1$ and $s_i = s_i + remtime(\mathrm{PTT})$.

So finally, Function **UpdateCatalog** $(\mathrm{T}, \mathrm{Q_{seq}}, \mathrm{Q_{multi}}, \mathrm{Q_{set}})$ is defined to update query catalogs with the new information contained in trace T.

Algorithm UpdateCatalog $(T_n, Q_{seq}, Q_{multi}, Q_{set})$
For every partial trace $PT_m \subseteq T_n$ (with $m \leq n$ and $m > 0$):
For every partial trace tail $PTT_h \subseteq PT_m$(with $h \leq m$ and $h > 0$):
Being i = **ExistsPartialTraceTail**$(\boldsymbol{PTT_h, Q_{seq}})$
if $(i \geq 0)$ **UpdatePartialTraceTail**$(\boldsymbol{PTT_h, Q_{seq}, i})$
else, **AddPartialTraceTail**$(\boldsymbol{PTT_h, Q_{seq}})$
Being i = **ExistsPartialTraceTail**$(\boldsymbol{PTT_h, Q_{multi}})$
if $(i \geq 0)$ **UpdatePartialTraceTail**$(\boldsymbol{PTT_h, Q_{multi}, i})$
else, **AddPartialTraceTail**$(\boldsymbol{PTT_h, Q_{multi}})$
Being i = **ExistsPartialTraceTail**$(\boldsymbol{PTT_h, Q_{set}})$
if $(i \geq 0)$ **UpdatePartialTraceTail**$(\boldsymbol{PTT_h, Q_{set}, i})$
else, **AddPartialTraceTail**$(\boldsymbol{PTT_h, Q_{set}})$

Algorithm 1: UpdateCatalog $(T_n, Q_{seq}, Q_{multi}, Q_{set})$

5.3 Remaining time estimations for a single partial trace tail

Let PTT_h be a partial trace tail with length h for which we want to calculate its remaining time. The estimation basically consists of comparing the partial trace tail PTT_h with its equivalent partial trace tail in the catalog, to calculate the average remaining time of the partial trace tails that matched PTT_h.

Now, the algorithm used to calculate the remaining time of a single partial trace tail PTT_h using one catalog is defined as following:

Algorithm CalculateRemTime(PTT_h, Q)
Output: The calculated remaining time of PTT_h using query catalog Q
Being i = ExistsPartialTraceTail(PTT_h, Q)
(1) if ($i \geq 0$) then **return** $\boldsymbol{s_i/q_i}$
(2) else, do the following:
(3) Being $PTT_h = \langle e_1, \cdots, e_h \rangle$, $PTT_{h-1} = \langle e_1, \cdots, e_{h-1} \rangle$ is defined as a partial trace tail equal to PTT_h, but without the last element e_h.
(4) $\Delta = timestamp(e_h) - timestamp(e_{h-1})$ is defined as the time difference between the last two events of PTT_h
(5) **return CalculateRemTime**($\boldsymbol{PTT_{h-1}, Q}$) $-$ Δ

Algorithm 2: CalculateRemTime(PTT_h, Q)

Steps (3), (4) and (5) are necessary when a partial trace tail PTT_h does not have an equivalent partial trace tail in the catalog, checked in (1). If this is the case, the algorithm takes a step back to the previous state of the partial trace tail (by removing the last event) until it finds at least one equivalent partial trace tail in the query catalog. Then it calculates the remaining time, compensating it for the steps it took back. This feature allows this algorithm to calculate the remaining time of partial trace tails that contains events that never had happened before, giving flexibility and robustness to this approach.

5.4 Remaining time estimations for a partial trace

Now that these definitions are made, the main issue that motivates this article will be covered. To calculate the remaining time of a partial trace, we have defined three new methods. These will be compared to the approach used in [4] and to a simple heuristic.

Simple heuristic method:

For a partial trace $PT_m = \langle e_1, \cdots, e_m \rangle$, the calculated remaining time with simple heuristic (SH) will be:

$$SH = \text{Max}\{\text{AverageCaseTime} - |\text{timestamp}(e_m) - \text{timestamp}(e_1)|, 0\}$$

Annotated transition system method:

This method, defined in [4], consists on the generation of a transition system and then annotates it with the time information contained in a training log. Then, partial traces are evaluated as it goes through the annotated transition system. The experi-

ments conducted with this method in this article will use only *set* as comparison type and a horizon of 1 event, to create, annotate and evaluate the transition system.

Notice that these parameters were the most common and successful parameters used in [4].

Proposed Method 1: Average catalog:

This method consists in evaluating a partial trace by estimating each remaining time for each partial trace tail possible contained in it, within each query catalog defined, then calculate the average of all of these estimations and return a single value.

Being $PT_m = \langle e_1, \cdots, e_m \rangle$ and $PTT_h \subseteq PT_m,\ h \leq m$, the calculated remaining time RT will be:

$$RT = \frac{1}{m} \sum_{h=1}^{m} \frac{\text{CalculateRemTime}(PTT_h, Q_{seq}) + \text{CalculateRemTime}(PTT_h, Q_{multi}) + \text{CalculateRemTime}(PTT_h, Q_{set})}{3}$$

The advantage of this method is that is less likely to over-fit the estimations than any method that uses only one match or equivalence.

Proposed Method 2: Best horizon and catalog:

This method uses recommendations about the *horizon* and *comparison type* by minimizing expected error, defined for each partial trace length. These optimal horizon and comparison types are calculated as follows:

The event log is separated into a training log and a test log. The training log is used to create the 3 query catalogs mentioned in 5.1 using the algorithms described in 5.2. Then the test log is used to calculate the remaining time of each PPT contained in the log and compare them to the real remaining time recorded in each PPT. So then, for all PPT that have the same length, one horizon and comparison type is selected: the one that has the minimum expected error, considering the differences between its calculated and real remaining times. So if the longest PPT in the training log has a length of n, for each length between 1 and n there is one recommended horizon and comparison type. So, the calculated recommended remaining time RRT for a PT_m will be: $RRT = \text{CalculateRemTime}(PTT_{h_m}, Q_{g_m})$, where PTT_{h_m} is the partial trace tail of length *m* and Q_{g_m} is the query catalog, and (h_m, g_m) are the horizon and catalog combination recommended for a partial trace length m.

Proposed Method 3: Default horizon and catalog:

To be able to make a fair comparison between annotated transition systems [4] and query catalogs, they must be executed using the same parameters (horizon and comparison type). In this case, the parameters will be the same as the annotated transition system method defined in 5.3.4: *set* as a comparison type and *1* as horizon. So, the calculated default remaining time DRT will be: $DRT = \text{Calculate}(PTT_1, Q_{set})$

5.5 Evaluating estimations

In order to determine the quality of the different estimation types, the log is divided into two parts; a training log and a test log. The training log is used to create the query catalogs, while the test log si used to calculate remaining times, and then comparing them to the real remaining times. To compare the methods defined above, 3 indicators were selected to measure a method's error (difference between calculated and real remaining times): Being $b^e{}_i$ be the remaining time estimation of an element i. Being

b^r_i be the real remaining time of an element i, and n be the number of elements analyzed:

Mean absolute error (MAE): Mean absolute error measures the average distance between both measurements.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |b^e{}_i - b^r{}_i|$$

Mean absolute percentage error (MAPE): Mean absolute percentage error measures the average percentage distance between both measurements.

$$\mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n} \frac{|b^e{}_i - b^r{}_i|}{b^r{}_i}$$

Root mean square error (RMSE): Rooted mean square error measures the root of the average squared distance between both measurements.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (b^e{}_i - b^r{}_i)^2}$$

MAE measures the real difference between the estimation and the real value; instead, MAPE considers the percentage difference, giving an equivalent significance to all measurements regardless of its absolute value. On the other hand, using RMSE is a good way to include measurement's variability into the analysis.

Cross validation [7] was used to homogenize results. In a K-fold cross-validation, sample data is split into K equal sized sub-groups. One of the sub groups is used as test data, while the rest (K-1 sub-groups) are used as training data. The cross validation procedure is repeated K times, each time using a different sub-group as test data. Finally, the arithmetic average of all results is calculated to obtain a single result. This procedure is very accurate, because it makes evaluations from K test and training data combinations. In practice, the number of iterations depends on the size of the original data set. The most common value used for K is 10.

To analyze the flexibility of the new methods introduced in this article, they were tested in two ways, each way will correspond to the same synthetic model; the first was to generate and analyze an event log that has no errors, the second was to generate an event log with 10% of its traces containing errors.

6 Experiments

To analyze the results of the methods described above, tests were run using different synthetic and real event logs.

In order to generate synthetic event logs, a software named Process Log Generator (PLG) [8] was used, which generates petri nets [9] from certain pattern probability defined by the user. Once the petri net has been generated, then the event log is created from it using simulation techniques. Several synthetic models were evaluated, but for a matter of length, only one synthetic model could be included in this article. The synthetic model used in the tests is shown as a petri net as follows:

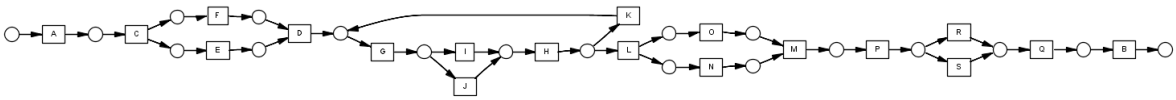

Figure 1: Synthetic model

This model is fairly simple, but contains most common process patterns: decisions, parallelisms and loops.

Log with no errors:

This log was generated based on the synthetic model above and it has 500 traces. Trace durations go from 202 to 432 hours, with an average of 272 hours and a standard deviation of 64 hours. The experiment results are as follows:

Method	MAE (hrs)	MAPE (%)	RMSE (hrs)
Simple Heuristic (SH)	52,97	52,11%	65,27
Annotated transition system	34,06	20,71%	47,72
Default horizon and catalog (DRT)	30,96	17,75%	**45,62**
Average catalog (RT)	30,91	17,75%	45,66
Best horizon and catalog (RRT)	**30,56**	**17,26%**	46,14

Table 1: Results for a synthetic event log without errors

The new methods and the annotated transition system seems to be better than the simple heuristic and the new methods outperform the annotated transition system, but the difference between the new methods and the annotated transition system is too low to make a conclusion.

Log with errors:

Then, we proceed to analyze the synthetic event log generated from the same synthetic model as the previous log, but this time having 10% of its traces with errors. An error in a trace represents any anomalous behavior that cannot be described by the petri net of the model it is based on, from changes in the order and number of events to new parallelisms, decisions or loops. This log has 500 traces. Trace durations go from 201 to 410 hours, with an average of 263 hours and a standard deviation of 49 hours. The experiments results are as follows:

Method	MAE (hrs)	MAPE (%)	RMSE (hrs)
Simple heuristic (SH)	40,60	42,57%	50,18
Annotated transition system	36,86	32,18%	50,77
Default horizon and catalog (DRT)	32,87	29,63%	46,17
Average catalog (RT)	**28,76**	**20,60%**	**40,35**
Best horizon and catalog (RRT)	37,98	28,70%	54,53

Table 2: Results for synthetic event log with errors

We can observe that when using all information available (average catalog method) the difference with all other methods is significant.

Moreover, we can observe that when using the best horizon and catalog method, the rooted mean squared error (RMSE) is higher than in all the other methods. This might be caused by the over-fitting of this method for a model that does not have a rigid structure (because of the incorporated errors). In this trace, all the information used by the average catalog method avoids over-fitting the model.

Real model log:

This log was extracted from a Chilean telecommunication company's systems. The analyzed process in this log is a "Technical issues resolution process", which consists in solving technical issues detected by customers, who call the company and request a solution for their issues.

This log has 261 traces. Its durations go from 3.3 hours to 58 days, with an average of 18 days and a standard deviation of 21 days. Notice that there is a **significant variability** in the data. The experiment results are as follows:

Method	MAE (days)	MAPE (%)	RMSE (days)
Simple heuristic (SH)	19,04	104,56%*	20,19
Annotated transition system	17,09	195,52%	19,06
Default horizon and catalog (DRT)	15,76	148,24%	**18,02**
Average catalog (RT)	**15,57**	**146,83%**	**18,02**
Best horizon and catalog (RRT)	15,83	154,05%	18,43

Table 3: Results from real event log

We can notice by looking only to the MAPE metric that the simple heuristic (*) has a better performance than all the other methods, but this is not the case, because when calculating the remaining time, the simple heuristic method presents irregularities when the trace elapsed time is more than the average cycle time, if so, the remaining time estimation will be 0. For example, if all remaining time estimations are always 0, MAPE will be 100%. This is even better than all the MAPE metrics calculated with the three other methods, but it does not make sense from a business or customer point of view. So in high variability scenarios is better to use MAE and RMSE to make conclusions. We can notice that, on high variability scenarios the average catalog method is better than all the other methods presented, but the results with all methods are not good enough from a business point of view, this might be caused by the high variability that exists in this process. These methods should be tested in more real-life logs to make better conclusions.

In this real process, the flexibility provided by the three new methods probably influenced the overall precision, giving them an advantage over previous existing methods. In real life, if any of these three new methods was implemented, new information could be added to the catalogs, and obsolete information could be removed, to maintain only the important information needed to estimate remaining times.

7 Conclusions

In this article we have introduced three new methods for calculating remaining time for partial traces, all of them based on a new approach based on the usage of query catalogs. In both synthetic and company-extracted real event logs, the new methods had better results than other existing methods, measured through three different error metrics. Then it is possible to conclude that these new methods based on query catalogs are more precise than the previous ones. The new approach based on query catalog usage not only allows improving estimation's precision, but also delivers more flexibility and dynamism, allowing to continuously improve estimations based on new information being collected. For example, we could determine the obsolescence of certain partial trace tails or the integration of new ones without needing to reprocess all the previous data, just deleting the obsolete information or adding the new information to the catalogs, both with low computational cost. We can observe that based on the results, the average catalog method behaves better with high variability processes, while the best horizon and catalog method behaves better with low variability processes. Nevertheless, more detailed experiments are required to make a better conclusion about which method is better in different circumstances.

8 Future work

As future work, we want to establish two parallel roadmaps: implementation and algorithm improvement. In the implementation roadmap, we are developing a plugin for the process mining tool ProM [10], to make this knowledge available to the process mining research community. In the algorithm improvement roadmap, we identify three improvement focuses. The first is about analyzing the behavior of the algorithms with different process patterns (decisions, parallelisms and loops), in order to adapt the strategies used for each one, thus improving the overall estimations. We think that a good starting point is to analyze process segments, allowing us to isolate patterns and analyze them individually, and then regroup results for calculating a single estimation. The second improvement focus is about analyzing the behavior of these methods in processes with different variability, in order to define the best method for each scenario. The third improvement focus is to dynamically select methods depending on the degree of advance of a trace; it is possible that when a trace is starting a method will be more adequate, while when it is close to ending, another method could behave better. Another interesting study subject is to analyze other possible catalog usage, e.g., to establish recommendations, namely show the most probable ways to go for the rest of the trace, and for each one calculate the remaining time estimation. This would allow us to use the query catalog approach to improve in a more comprehensive way actual recommendation systems.

9 References

1. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. (2005). Process-Aware Information Systems: Bridging People and Software through Process Technology. *Wiley & Sons.*
2. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. (2007). Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713-732.
3. H. Schonenberg, B. Weber, B.F. van Dongen, and W.M.P. van der Aalst. (2008). Supporting Flexible Processes Through Recommendations Based on History. In M. Dumas, M. Reichert, and M.C. Shan, editors, *International Conference on Business Process Management* (BPM 2008), volume 5240 of *Lecture Notes in Computer Science*, pages 51-66. Springer-Verlag, Berlin.
4. Van Der Aalst W.M.P., Schonenberg M.H., Song M. (2011). Time prediction based on process mining. *Information Systems*, 36 (2), pp. 450-475.
5. B.F. van Dongen, R.A. Crooy, and W.M.P. van der Aalst. (2008) Cycle Time Prediction:When Will This Trace Finally Be Finished? In R. Meersman and Z. Tari, editors,*Proceedings of the 16th International Conference on Cooperative Information Systems*, CoopIS 2008, OTM 2008, Part I, volume 5331 of *Lecture Notes in Computer Science*, pages 319–336. Springer-Verlag, Berlin.
6. Backus, P., Janakiram, M., Mowzoon, S., Runger, G. C., & Bhargava, A. (2006). Factory cycle time prediction with a data-mining approach. *IEEE Transactions on Semiconductor Manufacturing,* vol. 19, no. 2, pp. 252–258, 2006.
7. Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1137–1143).
8. Burattin, A., Sperduti, A.(2010): PLG: a Framework for the Generation of Business Process Models and their Execution Logs. In: *Proceedings of the 6th International Workshop on Business Process Intelligence* (BPI 2010). pp. 214–219. Springer Berlin Heidelberg, Hoboken, New Jersey, USA
9. Peterson, J.L. (1977). Petri Nets. *ACM Computing Surveys* (CSUR) 9(3), 223–252 (1977)
10. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. (2007) ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency* (ICATPN 2007), volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.