Enhancing Stochastic Petri Net-based Remaining Time Prediction using k-Nearest Neighbors

Jarne Vandenabeele[1], Gilles Vermaut[1], Jari Peeperkorn and Jochen De Weerdt

[1]*Co-first authors*

Research Center for Information Systems Engineering (LIRIS), KU Leuven, Leuven, Belgium

Abstract

Reliable remaining time prediction of ongoing business processes is a highly relevant topic. One example is order delivery, a key competitive factor in e.g. retailing as it is a main driver of customer satisfaction. For realising timely delivery, an accurate prediction of the remaining time of the delivery process is crucial. Within the field of process mining, a wide variety of remaining time prediction techniques have already been proposed. In this work, we extend remaining time prediction based on stochastic Petri nets with generally distributed transitions with k-nearest neighbors. The k-nearest neighbors algorithm is performed on simple vectors storing the time passed to complete previous activities. By only taking a subset of instances, a more representative and stable stochastic Petri Net is obtained, leading to more accurate time predictions. We discuss the technique and its basic implementation in Python and use different real world data sets to evaluate the predictive power of our extension. These experiments show clear advantages in combining both techniques with regard to predictive power.

Keywords

business processes, stochastic Petri nets, process mining, predictive process monitoring

1. Introduction

The application of Process-Aware Information Systems (PAIS)s, such as ERP, BPMS and CRM systems to support business processes is increasing [1]. These systems record information of the process execution and possibly the individual events of that process. Drawing insights and conclusions from these data is already possible using various process mining techniques categorized into process discovery, conformance checking, and extension [2]. Within the field of process mining, predictive process monitoring concerns itself with predictive techniques applied to process data. Predominantly, three types of predictions are considered as useful: next activity, outcome, and remaining time [3]. In this work we present a novel technique that combines a data-driven selection of candidate traces with building and simulating stochastic Petri nets, to predict the remaining time of running process instances. It builds upon the technique described by Rogge-Solti and Weske in [4, 5]. This technique, referred to as *generally distributed transition stochastic Petri net (GDT_SPN)*, tailors stochastic Petri nets to incorporate the time passed since the previous completed event. We show that by combining the flexibility of GDT_SPNs with a simple, yet effective, candidate selection approach, we can improve upon the accuracy of the

ALGORITHMS & THEORIES FOR THE ANALYSIS OF EVENT DATA 2022

jari.peeperkorn@kuleuven.be (J. Peeperkorn); jochen.deweerdt@kuleuven.be (J. De Weerdt)

predictions, as demonstrated experimentally on four real-life datasets. Our approach can be summarized as follows:

- Use the K-nearest-neighbor algorithm to identify the k instances most similar to the current prefix of this instance. The algorithm uses vectors based on the timestamps of previous events, while also taking into account the activity types.
- We use these k traces to discover a (new) Petri net using the Inductive Miner [6] and by performing a simulation a stochastic map is obtained which complements the Petri net, to eventually obtain a GDT_SPN.
- A simulation of this GDT_SPN is further used to estimate the remaining time. This is done n times and the actual prediction is taken to be the average of the n simulations.

The remainder of this paper is organised as follows. In Section 2, the most relevant related work is discussed. Section 3 defines some preliminaries, before Section 4 presents the core of our technique, i.e., how we predict the end time, and the basic implementation of it in Python. Section 5 evaluates our technique by comparing the results of different experiments with a number of benchmarks. We end this paper with Section 6, which summarises the paper, touches upon the limitations of our technique and mentions possible further enhancements.

2. Related Work

There already exist multiple techniques for predicting the remaining time of an ongoing process instance. For a comprehensive overview of the most relevant methods, interested readers are referred to Verenich et al. [7]. For the sake of conciseness this section is limited to an overview of those techniques that are either relevant for the vast majority of the methods discussed in [7], or those that are highly related to the framework discussed in this paper. Earlier work concerning remaining time prediction was proposed by van der Aalst et al. [8]. By applying finite state machine techniques on event logs, they learn an annotated transition system. Such a system extends a traditional transition system with predictive information by annotating measurements of time instances at each state. Two follow-up techniques are presented in [9, 10]. They enhance the technique of [8] by clustering the traces of the log in advance and creating an annotated transition system for each of these clusters afterwards. At runtime, a new trace will be assigned to a cluster and the annotated transition system for that cluster will then be used to predict the remaining time of that trace. A similar multi-stage approach is presented in this paper. The application of clustering to predictive business process monitoring has also been studied in other papers, e.g., [11].

Polato et al. [12, 1] present three approaches, all of them using Support Vector Regressors (SVR), to predict the remaining time starting from a certain state. The first two are based on regression techniques with or without control-flow information, where the event log that serves as input is initially transformed in order to be suitable for the ε-SVR algorithm. The last approach is again mainly based on the idea of annotated transition systems as mentioned above [8]. It enriches each state with a Naive Bayes classifier and each transition with a Support Vector Regressor, yielding a Data-Aware Transition System (DATS).

The above mentioned techniques are often based on support vector machines and/or regression. These are, however, not the only machine learning techniques used to create predictive

models. For instance, regression trees [13], XGBoost [14], or random forests [15] have already been applied for remaining time prediction. The increasing interest and latest achievements in deep learning techniques have led to a rise in predictive models using recurrent neural networks, convolutional neural networks, generative adversarial nets and, more recently, transformer nets, together with multistage approaches [16, 17, 18, 19, 20, 21, 22, 23, 24]. While these works show promising results, the presented models are so-called *black box*, i.e. their results are hard or even impossible to interpret. This limits their use in applications where the explainable aspect is key, e.g. root-cause analysis.

We particularly highlight the work of Rogge-Solti and Weske [4, 5], as their approach has been the main inspiration for the technique presented in this paper. Their GDT_SPN-formalism based technique differs from other approaches as it takes into account the time passed since the last observed event, while the other approaches only update predictions upon arrival of finished events. To achieve this, stochastic Petri nets are equipped with generally distributed transitions, in which distributions reflect the duration of the corresponding activities in the real world and can be different from the exponential distribution, i.e., the Markovian property is not enforced. The latter distributions may then be updated, based on the time passed since the last observed event, to achieve more accurate predictions. These models are not *black box*, and can thus be used to explain multiple aspects influencing the execution time of a certain instance.

3. Preliminaries

In this chapter, we describe the preliminary concepts on which our novel prediction technique is based. As mentioned above, the goal of this technique is to predict the remaining time of ongoing cases. This entails that the predictive model can only use information of the current case up until this point in time, supplemented with information from past cases. The assumption is made that traces contain timestamps for each occurring event. Those timestamps indicate at least the end, and in some cases the start, of the activities represented in the trace. Each event in the event log should contain a *trace ID*, indicating to which process execution it belongs, together with an activity type. Our technique is an extension of the work of Rogge-Solti and Weske, who introduce the use of stochastic Petri nets with generally distributed transitions, so-called GDT_SPN models, to make predictions [4, 5]. In this technique, a Petri net is enriched with statistical timing data for each event, which allows end-users to make time predictions. The definition of both a Petri net and a GDT_SPN as presented by Rogge-Solti and Weske is given below [5]:

Definition 1 (Petri Net) A Petri net is a tuple PN = (P, Tr, F, M_0) where:

- P is the set of places.
- Tr is the set of transitions.
- $F \subseteq (P \times Tr) \cup (Tr \times P)$ is the flow relation.
- $M_0 \in P \rightarrow \mathbb{N}_0$ is the initial marking.

The Petri net models used by [5], and hence as well in our technique, are restricted to sound workflow nets.

Definition 2 (Generally Distributed Transition Stochastic Petri Net) A GDT_SPN is a seven-tuple: GDT_SPN = (P, Tr, $\mathscr{P}$, $\mathscr{W}$, F, M_0, $\mathscr{D}$), where (P, Tr, F, M_0) is the basic underlying Petri net as specified above. Additionally:

- The set of transitions Tr is split into immediate transitions Tr_i and timed transitions Tr_t.
- $\mathscr{P}$: Tr $\rightarrow \mathbb{N}_0$ is the assignment of priorities to the different transitions, where $\forall \tau \in Tr_i$: $\mathscr{P}(\tau) \geq 1$ and $\forall \tau \in Tr_t$: $\mathscr{P}(\tau) = 0$.
- $\mathscr{W}$: $Tr_i \rightarrow \mathbb{R}^+$ assigns probabilistic weights to the immediate transitions Tr_i.
- $\mathscr{D}$: $Tr_t \rightarrow$ D is the assignment of probability distribution functions D to timed transitions Tr_t, reflecting the durations of the corresponding activities.

These probability distribution functions D do not need to be exponentially distributed, but can also be normal distributions, uniform distributions, etc. and hence do not necessarily have the Markovian property, i.e., memorylessness. This is an important factor as the absence of this property is used to update the distribution functions based on the passed time. This is the key difference with the more known generalised stochastic Petri nets (GSPN), as presented by Marsan et al. [16]

Rogge-Solti and Weske [4, 5] exploit the idea of memorylessness to update the density function of the original distribution towards a new density function of a *truncated* distribution. The intuition behind this is that when some time has passed, the probability increases that the activity will be completed in the nearer future as some work might already have been done. This is in contrast with the above mentioned Markovian property. This is done by using the elapsed time since the last event. Let $F_\delta(t)$ be the duration distribution function of that activity. By differentiating $F_\delta(t)$ you can obtain a density function $f_\delta(t)$. We then use t_0, the current time since enabling the transition, to truncate the distribution as follows [5]:

$$f_\delta(t|t \geq t_0) = \begin{cases} 0 & t < t_0, F_\delta(t_0) < 1 \\ \frac{f_\delta(t)}{1-F_\delta(t_0)} & t \geq t_0, F_\delta(t_0) < 1 \\ f_{\delta_{Dirac}}(t - t_0) & F_\delta(t_0) = 1 \end{cases}$$

The part of the density function above (or in this case more correctly *after*) t_0 is rescaled in such a way that it integrates to 1. For the case where $F_\delta(t_0) = 1$, i.e. when the current time has progressed further than the density functions supports, we use the Dirac delta function $f_{\delta_{Dirac}}$ (a function whose value is 0 everywhere except at one peak, and whose full integral is equal to 1), with a peak at t_0. This happens when the current event is taking longer to complete than the events in the training log corresponding to the same activity type. The basic idea is that the predictions are only based on those cases for whom the corresponding activity would not have been already completed at the current time t_0. For a more extensive explanation, together with the effects of this truncation of different types of distributions, interested readers are referred to [5]. In addition, n is the number of simulations performed by the GDT_SPN, for a given sample case. The eventual prediction is equal to the mean duration. As will be explained in Section 4, our proposed prediction technique combines the algorithm as described by [4, 5] with the basic idea of the kNN algorithm. We have adopted the notion of finding the k most representative training instances for the to-be-predicted instance. These k representative cases found during kNN are used to build a predictive GDT_SPN model that can then be used to make a prediction.

4. Remaining Time Prediction using GDT_SPN_*k*NN

In this Section, we introduce generally distributed transition stochastic Petri net with *k*NN-based candidate selection, referred to as GDT_SPN_*k*NN. The algorithm depends on the number of neighbors taken into account, hyperparameter k. In order to make predictions, the algorithm further requires a log file containing complete traces of the business process (the *training log*), the time passed since the start of the case t_0, and a partial trace T, containing events with timestamps until t_0. The key extension of GDT_SPN_*k*NN with respect to the original GDT_SPN algorithm, resides in the fact that we avoid using the whole training log as an input to construct the GDT_SPN model. In contrast, in GDT_SPN_*k*NN, only the k complete traces that are most similar to the to-be-predicted partial trace T are taken into account. The main motivation for selecting this subset is that, under the condition that the right features are selected, only looking at the k most similar traces will yield a predictive model that embodies less variation in the generalised density functions, likely leading to a final prediction closer to the real value. If tuned well, selecting only the k nearest instances thus omits irrelevant cases and outliers and yields a more representative and stable GDT_SPN model for the given partial trace.

The choice of the hyperparameter k to decide the number of nearest neighbors is not obvious. When k is taken too small, the event distributions built by the k neighbors may be inaccurate or have a high variance, which leads to volatile and most of the time worse results. When k is taken too large, the effect of looking at only the most similar traces may be minimal or even non-existent, given that the larger k, the more the outcomes will look like those of the original technique. By taking k too large, the variance may increase as well, as multiple trace variants are taken into account for building the model, which leads to a more complex Petri net. These different trace variants may also have a different underlying distribution to estimate the duration of the activities. Another drawback of setting k too large is that the computation time increases, as more traces need to be replayed to construct the Petri net and distributions. Nonetheless, despite the fact that hyperparameter k is key, it is computationally demanding to tune it. Based on initial explorations, we found that a value of 100 provided to be a good trade-off value. While in a practical application, it would be certainly beneficial to tune k carefully, in this paper, mainly due to a lack of time, we work with this fixed value of 100. In further research, we would like to investigate dedicated strategies to tune k, beyond an exhaustive search.

For applying k nearest neighbors, a proper featurization should be applied. While a conventional feature engineering would typically consider trace and event attributes, we propose a method that only relies on time information. More specifically, on a per event basis, we take into account the total time from the start of each trace t_0 until the occurrence of that individual event. This total duration between the start of the *trace* and the occurrence of that *event* will be referred to as the *time-to-occurrence* of an event in the remainder of this paper. In case of the presence of loops, the *time-to-occurrence* is determined based on the last repeated activity observed in a trace. The rationale behind using *time-to-occurrence* is that there is a likely correlation between traces that have a similar time-to-occurrence for events corresponding to the same activity type and hence, it might thus be valuable to sample them in order to make better remaining time predictions. This feature engineering is also more generalisable compared to matching neighbors based on attributes, as not all log files contain event and/or trace attributes. However, for certain processes, it might well be that taking into account other trace or event features is

beneficial. Our algorithm easily allows to incorporate this, when e.g. expert knowledge suggests correlations between these features and the remaining time. When we eventually build the GDT_SPN using the kNN, we still do n different simulations for which we take the average values as our eventual prediction. In this work, the value of $n = 500$ is chosen, adopted from [5], as taking this high enough will make the predictions more robust and less volatile.

Algorithm 1: Feature Construction of the times_to_occurrence vector for a trace

Result: Times_to_occurrence vector ψ^* for a trace ψ
ψ = Trace ;
Voc = [all activity types in training data];
Function *Trace_times_to_occurrence(ψ, Voc)*
  ψ_{start} = start_time(ψ);
  $\psi^* = [\psi_1^*, \psi_2^*, \dots, \psi_D^*]$ with $D = |\text{Voc}|$;
  for $i \in \{1, \dots, D\}$ **do**
    if $Voc_i \in activities(\psi)$ **then**
      event = event corresponding to the last occurring event of activity type Voc_i;
      endtime = timestamp(event);
      ψ_i^* = endtime - ψ_{start};
    else
      $\psi_i^* = -1$;
    end
  end
  return ψ^*
end
Function *activities(ψ)*
  Intitialize: $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_L]$ with $L = |\psi|$;
  for $i \in \{1, \dots, |\psi|\}$ **do**
    α_i = Activity type of event ψ_i
  end
  return α
end

The feature construction needed to predict the remaining time of a (partial) test trace T can be seen in Algorithm 1. We define an activity vocabulary *Voc* containing all possible activity types present in the process. Each trace ψ of the training log is formatted as follows. For each activity present in the activity vocabulary of the event log, the algorithm checks whether an event occurs in ψ corresponding to that activity type. If it does, then the difference between the time of this event in ψ and the beginning of this trace is calculated and stored in a formatted vector ψ^* corresponding to this trace ψ. If it does not, a negative value is assigned. We call this value the *time-to-occurrence* of that activity type (ψ_i^* in the algorithm above). For the to-be-predicted trace T a similar formatted vector is constructed. If a certain activity type is present in the trace multiple times, as is the case for e.g. loops, we only consider the last occurring event. We choose to take the last occurrence, because if a certain activity has to be performed multiple times (in one execution), we assume most often the final completion time of that activity (i.e. the timestamp of the last occurrence) provides the most useful information. However, this choice can be easily altered if needed.

The result of this procedure is that for each trace, we obtain a formatted counterpart in the form of a vector with a fixed length. The length corresponds to the number of activities in the

business process, i.e., all distinct activities observed in the training data. Each entry in this vector corresponds to a fixed activity. The entry is positive if an event corresponding with the completion of the activity appears in the trace, and is negative otherwise. For the remainder of this explanation, we call the formatted version of the to-be-predicted trace T^*. Only the distance between relevant events is measured. These are the events that appear in the partial test trace T and consequently have a positive value in T^*. We want traces from the training log that follow the same path, i.e. having the same executed activities, as the partial trace to have a higher impact, since these traces are more likely to be similar to the partial trace in question. Accordingly, a penalty, in the form of a maximal distance vector, is induced on those that do not follow the same route up to the point of prediction as the partial test trace. This means that the assigned formatted vector for such a trace will contain a large constant for each event, i.e., the maximum time-to-occurrence seen in the training set, making this trace very far off when selecting the nearest neighbors. A min-max normalisation is applied on the formatted versions of the training traces and on T^*, which gives all events equal influence in calculating the distance. This makes them suitable for the kNN algorithm. If k is higher than the number of traces present in the training set that follow the same route, the algorithm randomly selects the other traces up to k. In a future extension, this might be replaced by taking prefixes closest to the control-flow of prefix T, by some metric. We use the Euclidean distance between the formatted vectors as a distance function in the kNN algorithm. In summary the executed procedure has the following characteristics:

- Only the distance between relevant events is measured. These are the events that appear in the partial test trace T and consequently have a positive value in T^*. The formatted versions of the training traces can have positive values for other events as well, but these are not taken into consideration when selecting the nearest neighbors, as only the events present in T are used to calculate the distances.
- We want traces from the training log that share all relevant events with the partial trace to have a higher impact, since these traces are more likely to be similar to the partial trace in question. That is why a penalty is induced on those that do not follow the same route up to the point of prediction as the partial test trace. It should be noted that the rare situation may occur where the number of kNN is higher than the number of traces present in the training set that follow the same route. If this situation happens, the algorithm will randomly pick the other traces up to k. This is another motivation for proper parameter tuning and to keep the parameter k small enough.
- Because of the normalisation of the vectors, one should remark that all completed events have an equal influence while calculating the distance. In the case an outlier is present, the distance between normal traces will be rather small.
- An important consequence of taking the nearest neighbors to build our model is that we allow the process to be dynamic. The occurrence of a new trace variant in the business process will be fully taken into account in the model as soon as k training examples of this trace variant are recorded.

When the nearest neighbors are found, the full training traces corresponding to these neighbors are used to create a Petri net using Inductive Miner [6], which guarantees to produce sound

and fitting models and hence alleviating some of the shortcomings of the α-miner [25]. As mentioned in Section 3, this soundness is a necessity for the Rogge-Solti and Weske algorithm [4, 5]. Additionally, fitting models make it possible to replay the partial trace T, which is necessary for the algorithm to make a prediction [26]. Given this Petri net and k training traces, simulation can be performed to obtain a stochastic map to complement the Petri net. A stochastic map projects every activity on a probability distribution function. In this way, we obtain a GDT_SPN that can be used for prediction using the original simulation of Rogge-Solti and Weske. Here we have to set another hyperparameter n, which is equal to th number of simulations we perform for the partial trace. The actual prediction is then taken as the average of all these simulations. For the rest of this work we use n equal to 500, which should be high enough to average out outliers in the simulation.

We have translated the implementation of Rogge-Solti and Weske [4, 5], that was available in the open-source program ProM, to a standalone implementation in Python compatible with the *pm4py* framework [27]. It does not yet fully cover all features that are available in ProM, but it covers the core algorithm and achieves similar results. Our standalone implementation, including the addition of the kNN algorithm, can be found on Github[1]. For the kNN algorithm we use the implementation in Scikit-Learn [28].

Our approach is agnostic to the specific candidate selection-technique, given that kNN could be replaced by another technique. For instance, this might be an eager clustering technique where for each cluster a GDT_SPN model can be constructed during the training phase. Similar approaches have been explored in [9, 10, 11]. One advantage of taking such an approach is that, as opposed to the lazy learner kNN, this would yield a better performance in the deployment phase. However, there are also reasons why using kNN could result in more accurate predictions. Eager clustering techniques often yield large sized clusters in addition to some smaller clusters, which takes away the power of combining smart candidate selection with the approach of [4, 5]. Additionally, using kNN allows the process to be dynamic, as changes can be picked up quickly in the resulting GDT_SPN model, leading to more accurate predictions. Depending on the importance of stressing either performance or accuracy and flexibility, one can choose an appropriate kind of learner when adapting our approach.

5. Experimental Evaluation

5.1. Experimental setup

In this section, we will discuss the results of our approach on four real-life datasets. A schematic overview of the setup is given in Figure 1.

Each real-life dataset is split into a training and a test log. In order to compromise between a large enough test set and reasonable computing time, all experiments use a test log with approximately 500 traces, independent of the amount of traces in the training log. The size of the training log is always significantly larger than the size of the test log, ranging from 2500 to 8000 traces, depending on the size of the original dataset from which we took a subset. The train and test log split was done out-of-time, in order to avoid possible data leakage. The training

[1] https://github.com/JarneVDB/BP-Time-Prediction-using-KNN

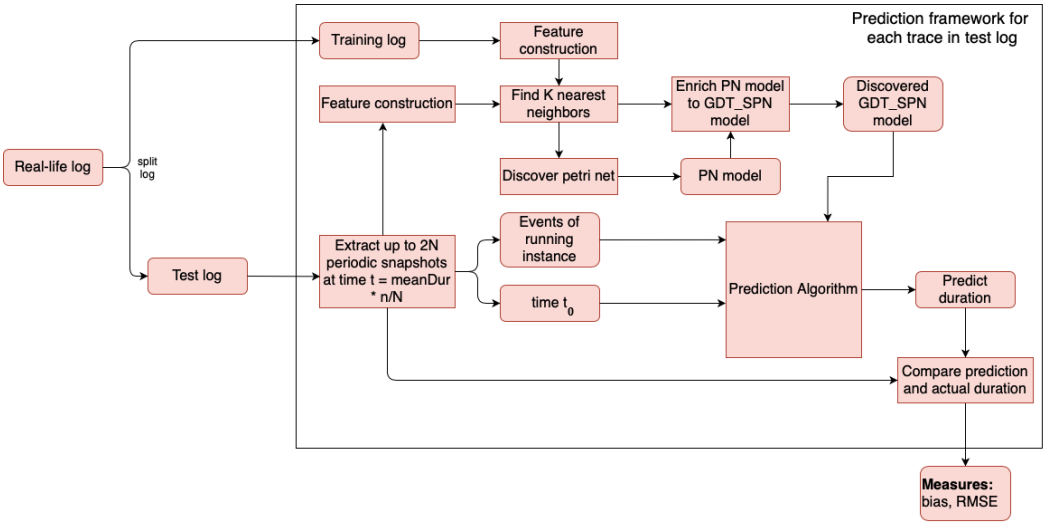

Figure 1: Evaluation setup

log used to obtain the k neighbors only contains finished traces upon to that point in time. All following steps are repeated x times, with x corresponding to the number of traces in the test log. For each trace in the test log, at most $2N$ periodic prediction iterations will be performed, where N is a hyperparameter that influences the number of prediction evaluation iterations. Corresponding to the methodology of [5], the Nth prediction will be performed *meanDur* after the start of the case, where *meanDur* corresponds to the average case duration of the training log. For all experiments, we set N equal to 20, leading to 40 prediction iterations. Each iteration thus refers to a point in time after the start of the trace, where we predict the remaining item of this execution. 2N can thus be regarded as the number of different points in time where we decide to predict the remaining lead time. Each time we do perform a prediction we provide the algorithm with the current traces up to t_0 (corresponding to this point in time). We call the specific point in time we are calculating the remaining times of all (still) ongoing cases, the prediction iteration. Hence, at most 40 moments of prediction are simulated for each trace, each corresponding to different stages in the execution of the process instance. When the execution has finished, at some iteration, no more predictions are done for this particular trace, thus most of the cases in the test log will only influence part of the prediction iterations (as they will be finished at some point). It should be noted that when the process has a low variance, the effective number of prediction iterations for each test trace will be close to N. When increasing the iteration, less and less traces are used to evaluate the time prediction, making the the results more volatile and statistically less significant.

For both the partial test trace T and the entire training log, feature construction is executed as described in Section 4, yielding the corresponding vector representations with the times-to-occurrence of all known activity types. In the transformed version of the training log, the 100 nearest neighbors of the formatted test trace T^* are found. The full traces corresponding to these neighbors are used to discover a Petri net, which is enriched to a GDT_SPN model

by means of stochastic information resulting from a simulation. Although the algorithm is flexible in the choice of distribution types, we decided to force a normal distribution for all non-immediate events. The choice of the most proper distribution is most liekly event log (process) dependent, but the normal distribution was chosen for every process in this paper due to time constraints. A further investigation on this topic, might clarify certain (possible) issues.

The results of the predictions are better when forcing normal distributions, given that the effect of the model vanishes when using memoryless exponential distributions and alternative distributions such as the uniform distribution are sensitive to outliers. Moreover, we assume that the normal distribution is the distribution type that is often a good estimation for the real underlying distribution function. This GDT_SPN model, together with the events of the running instance T and time t_0, serve as input for the prediction algorithm as described in [4, 5]. The number of different simulations performed by each GDT_SPN n, of which the purpose is explained above, is set to 500 for each experiment as this averages out most of the influence of outliers. The reported evaluation metrics will be the average error and the root mean square error (RMSE). These two metrics will allow us to respectively measure the bias and accuracy of our prediction method. Four different event logs are used in the experimentation. The first one, as a simple proof-of-concept, uses the BPI Challenge 2019 Event Log. However, instead of using the full event log, one single control-flow-variant is selected, which can be seen in Figure 2. The experiments on the other selected event logs, do use multiple different control-flow variants. These event logs are the *Hospital log*, depicting the billing process in a hospital, and the BPI Challenge 2020 event logs, depicting the travel expense declaration process of a university for domestic (*BPI2020d*) and international travel (*BPI2020i*). An overview of summary statistics regarding the different data sets can be found in Table 1.

Datasets	Cases	Events	Event classes	Max case length	Avg. case length	Max case time	Avg. case time
BPI2019[2] (1 variant)	7,460	44,760	6	6	6	356.21	94.54
Hospital[3]	7,847	33,450	7	6	4.26	867.54	156.95
BPI2020d[4]	7,820	40,281	7	6	5.15	290.89	10.52
BPI2020i [5]	2,361	23,726	14	12	10.05	463.04	80.17

Table 1
Descriptive statistics of event logs used for the training of the model. Time-related characteristics are reported in days.

[2]https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1
[3]https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741
[4]https://doi.org/10.4121/uuid:3f422315-ed9d-4882-891f-e180b5b4feb5
[5]https://doi.org/10.4121/uuid:2bbf8f6a-fc50-48eb-aa9e-c4ea5ef7e8c5

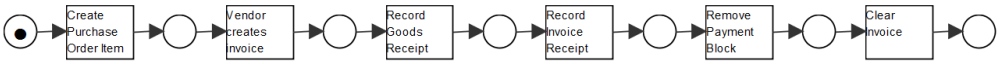

Figure 2: Petri net of the selected control-flow variant in the BPI Challenge 2019 dataset, illustrating a purchase order business process.

5.2. Experiment Results

GDT_SPN_*k*NN is tested against four benchmarks: the average duration of the full training set (Average), the average duration of the ten nearest neighbors of the candidate trace's prefix (Average 10 *k*NN), the average duration of the hundred nearest neighbors (Average 100 *k*NN), and the algorithm as proposed by Rogge-Solti and Weske (GDT_SPN) [4, 5]. One should note that for later iterations, only few to-be-predicted traces remain in the test set and that the results can thus be volatile and less informative towards the very end. The outcome of these experiments is visualised in Figure 3 and Figure 4, where the former reports the mean errors and the latter reports the root mean square errors. Note that in these figures our prediction algorithm is referred to as *GDT_SPN_kNN*. The x-axis represents the number of prediction iterations (as explained above), while the y-axis expresses the value of the corresponding metric in seconds.

In Figure 3 and Figure 4 no systematic under- or overestimation is observed. The bias compared to the benchmarks becomes smaller as more information about activities is known, i.e., as the prediction iteration goes up. In the vast majority of the prediction iterations, we achieve a higher accuracy than all benchmarks. The better predictions can be explained by two main factors. First, when there exists correlation between the times-to-occurrence of the different events, the algorithm can exploit this as soon as the first time-to-occurrence becomes available.

A second factor is that the Petri net for each cluster is more simple since it uses fewer trace variants, whereas the original GDT_SPN constructs a Petri net out of all training traces. This has a consequence in the further simulation of the Petri net as it is more likely that the ending is fixed, while in the more complex Petri net of [4, 5], multiple paths can be followed towards the final marking that actually belong to other trace variants. These trace variants might contain events portraying activity types not present in the test trace and might therefore yield worse predictions. Whenever the different trace variants have some matching activities, this factor becomes particularly important as the further simulation of the complete Petri net as in [4, 5] would be too volatile. It should be noted that although the results on the above data sets, on average, are showing significant improvements with regard to the stated benchmarks, there are some cases in which our approach does not yield better results. The predictions taken earlier on, at a lower iteration, show less difference between the tested methods as well. The benchmarks using simple averaging score not far off from the more elaborate GDT_SPN based methods. Later in the traces (for a higher iteration), the advantage of incorporating the time already passed on an activity, and possibly other information concerning the activities yet to come, result in more accurate predictions.

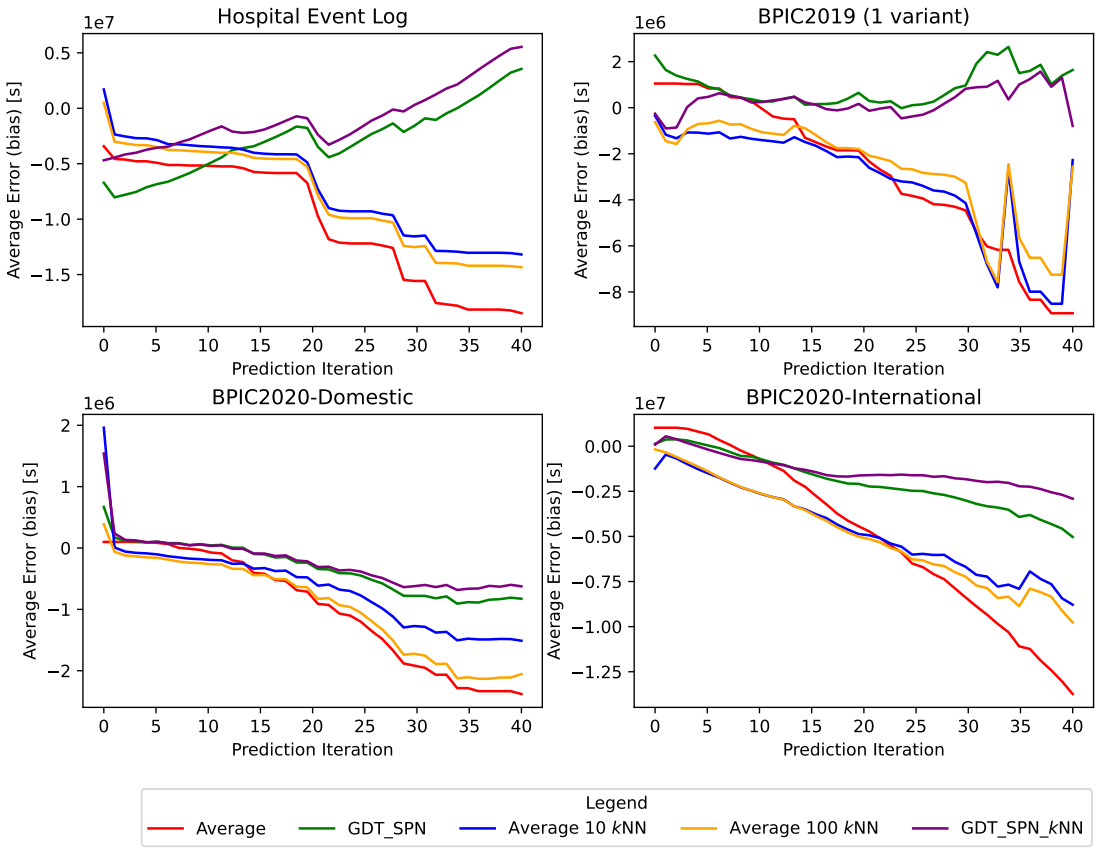

Figure 3: Mean errors of real-life experiments.

6. Conclusion and future work

In this paper, we constructed an approach for predicting the remaining time of a business process based on a combination of nearest neighbor selection and the GDT_SPN model as presented in [4, 5]. According to the four datasets we used to validate our model, we significantly outperform our four benchmarks, including the original method [4, 5]. The higher the correlation between the different times-to-occurrence and the more path variants with similar activities, the better our algorithm performs compared to the benchmarks. The GDT_SPN model itself is *white box*, and can therefore be used for explainability purposes. And while the kNN selection adds some complexity to the methodology, this does not obstruct explainability, and might even improve it when the discovered Petri nets are more simple.

Multiple assumptions were made in the experiments presented in this paper, such as putting the number of neighbors $k = 100$. The impact of these choices could be investigated further. Next to this, there are still multiple ways to build further on the prediction method presented in this paper, as both the choice of distance metric, clustering method and even the choice of using

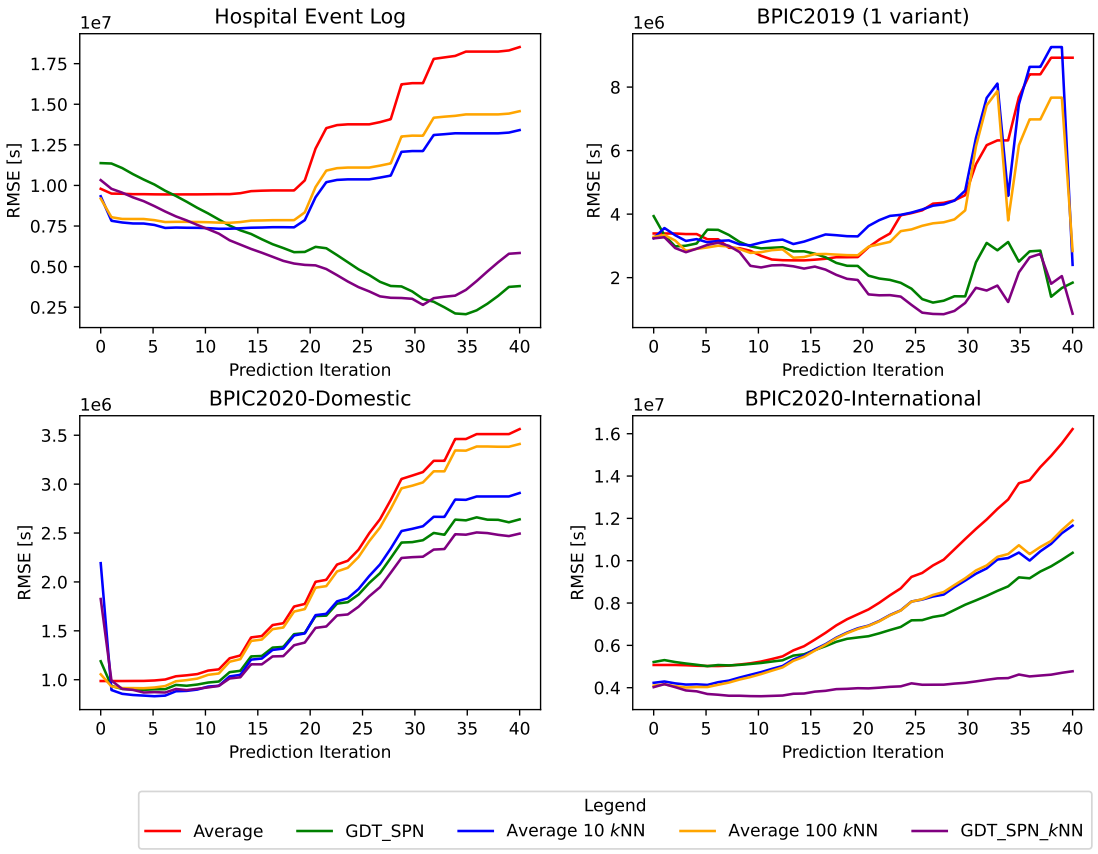

Figure 4: Root mean square errors of real-life experiments.

a GDT_SPN instead of something something else, is flexible and can easily be interchanged. One could develop a way to adjust the weighting of the activities by putting more weight on more relevant activities. This could potentially lead to better neighbor selection and hence better prediction. One could take into account trace and event variables while selecting nearest neighbors. A possible way of doing this is by a nested clustering approach, were one first clusters on either attributes or times-to-occurrence and afterwards reduces the number of neighbors by clustering on the other one. Furthermore, these extra attributes could be used to select the most relevant prefixes, when less than k prefixes can be found whose control-flow correspond to that of the case in question. In order to improve scalability, one could create fixed clusters and assign each test trace to the cluster it is most similar to. With such an eager clustering approach instead of the kNN algorithm, each cluster would have its own GDT_SPN model. These models can be built upfront and can directly be used to make a prediction from the moment the test trace is assigned to the corresponding cluster. While this may increase performance, the accuracy may drop, especially when the process is dynamic. Moreover more

experimentation into the impact and sensitivity of the results with different parameter values when setting up the inductive miner and different ways of matching neighbors, could provide some interesting insights. Moreover, investigating the true duration distributions might be interesting, as in this work we assumed Normal distributions. If the true distribution is not normal, learning different kinds of parametric (or even non-parametric) models might increase the prediction accuracy.

References

[1] M. Polato, A. Sperduti, A. Burattin, M. de Leoni, Time and activity sequence prediction of business process instances, Computing 100 (2018) 1005–1031.

[2] W. M. P. van der Aalst, Process Mining, Communications of the ACM. 55 (2012) 76–83.

[3] A. E. Marquez-Chamorro, M. Resinas, A. Ruiz-Cortes, Predictive monitoring of business processes: A survey, IEEE Transactions on Services Computing 11 (2018) 962–977.

[4] A. Rogge-Solti, M. Weske, Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8274 LNCS (2013) 389–403.

[5] A. Rogge-Solti, M. Weske, Prediction of business process durations using non-Markovian stochastic Petri nets, Information Systems 54 (2015) 1–14.

[6] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering block-structured process models from incomplete event logs, in: G. Ciardo, E. Kindler (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2014, pp. 91–110.

[7] I. Verenich, M. Dumas, M. La Rosa, F. M. Maggi, I. Teinemaa, Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring, ACM Transactions on Intelligent Systems and Technology 10 (2019).

[8] W. M. P. van der Aalst, M. H. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2011) 450–475.

[9] F. Folino, M. Guarascio, L. Pontieri, Discovering context-aware models for predicting business process performances, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7565 LNCS (2012) 287–304.

[10] F. Folino, M. Guarascio, L. Pontieri, Discovering high-level performance models for ticket resolution processes, in: R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. De Leenheer, D. Dou (Eds.), On the Move to Meaningful Internet Systems: OTM 2013 Conferences, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 275–282.

[11] C. Di Francescomarino, M. Dumas, F. M. Maggi, I. Teinemaa, Clustering-Based Predictive Process Monitoring, IEEE Transactions on Services Computing 12 (2019) 896–909.

[12] M. Polato, A. Sperduti, A. Burattin, M. de Leoni, Data-aware remaining time prediction of business process instances, Proceedings of the International Joint Conference on Neural Networks (2014) 816–823.

[13] M. de Leoni, W. M. van der Aalst, M. Dees, A general process mining framework for

correlating, predicting and clustering dynamic behavior based on event logs, Information Systems 56 (2016) 235–257. URL: https://www.sciencedirect.com/science/article/pii/S0306437915001313. doi:https://doi.org/10.1016/j.is.2015.07.003.

[14] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina, F. M. Maggi, Intra and inter-case features in predictive process monitoring: A tale of two dimensions, Lecture Notes in Computer Science 10445 LNCS (2017) 306–323.

[15] S. V. D. Spoel, M. V. Keulen, C. Amrit, LNBIP 162 - Process Prediction in Noisy Data Sets: A Case Study in a Dutch Hospital, International Symposium on Data-Driven Process Discovery and Analysis (2013) 60–83.

[16] M. Ajmone Marsan, G. Conte, G. Balbo, A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems, ACM Transactions on Computer Systems (TOCS) 2 (1984) 93–122.

[17] M. Camargo, M. Dumas, O. González-Rojas, Learning Accurate LSTM Models of Business Processes, Lecture Notes in Computer Science 11675 LNCS (2019) 286–302.

[18] J. Evermann, J.-R. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decision Support Systems 100 (2017) 129–140.

[19] N. Tax, I. Verenich, M. La Rosa, M. Dumas, Predictive business process monitoring with lstm neural networks, Lecture Notes in Computer Science (2017) 477–492.

[20] N. Mehdiyev, J. Evermann, P. Fettke, A multi-stage deep learning approach for business process event prediction, in: 2017 IEEE 19th Conference on Business Informatics (CBI), volume 01, 2017, pp. 119–128. doi:10.1109/CBI.2017.46.

[21] L. Lin, L. Wen, J. Wang, MM-Pred: A Deep Predictive Model for Multi-attribute Event Sequence, 2019, pp. 118–126. doi:10.1137/1.9781611975673.14.

[22] V. Pasquadibisceglie, A. Appice, G. Castellano, D. Malerba, Using convolutional neural networks for predictive process analytics, in: 2019 International Conference on Process Mining (ICPM), 2019, pp. 129–136. doi:10.1109/ICPM.2019.00028.

[23] F. Taymouri, M. L. Rosa, S. Erfani, Z. D. Bozorgi, I. Verenich, Predictive business process monitoring via generative adversarial nets: The case of next event prediction, in: D. Fahland, C. Ghidini, J. Becker, M. Dumas (Eds.), Business Process Management, Springer International Publishing, Cham, 2020, pp. 237–256.

[24] Z. A. Bukhsh, A. Saeed, R. M. Dikman, Processtransformer: Predictive business process monitoring with transformer network, CoRR abs/2104.00721 (2021). URL: https://arxiv.org/abs/2104.00721. arXiv:2104.00721.

[25] S. J. Leemans, D. Fahland, W. M. Van Der Aalst, Discovering block-structured process models from event logs - A constructive approach, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7927 LNCS (2013) 311–329.

[26] J. C. Buijs, B. F. Van Dongen, W. M. P. Van Der Aalst, Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity, International Journal of Cooperative Information Systems 23 (2014) 1–18.

[27] A. Berti, S. J. van Zelst, W. M. P. van der Aalst, Process mining for python (pm4py): Bridging the gap between process- and data science, CoRR abs/1905.06169 (2019). URL: http://arxiv.org/abs/1905.06169. arXiv:1905.06169.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel,

P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.