Comparing strategies to generate experience-based clinical process recommendations that leverage similarity to historic data

Steven Mertens, Frederik Gailly, Geert Poels
Faculty of Economics and Business Administration
Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium
{steven.mertens,frederik.gailly,geert.poels}@ugent.be

Abstract—Doctors executing flexible and knowledge-intensive healthcare processes are constantly confronted with decisions on what to do next from a large and diverse set of options. Such decision making can be very demanding and even overwhelming, certainly for less experienced doctors. The current IT infrastructure remains lacking in support for decision making. Adding an additional channel for externalizing the knowledge and experience from historic data as guidance would have clear benefits. In this paper, we take the first step towards offering such guidance in the form of clinical process recommendations for the next activity to be executed. The purpose of these recommendations is not to steer the process according to some optimization criteria, but rather to offer a direct reflection of the experience encapsulated in previous executions of the process. Forty strategies were defined to calculate probability estimates for each possible next activity. The recommendations are subsequently generated as a list of possible next activities, sorted according to the highest calculated probability. The strategies were implemented and applied to three different healthcare data sets and evaluated on average accuracy, multi-class brier score, log loss, how consistent the recommendation rankings are, and the required computation time. The results indicate that responded frequency, variable-position activity similarity and combined strategies perform well for this type of processes.

Keywords—Experience Externalization, Flexible and Knowledge-Intensive Processes, Next-Activity Process Recommendations, Trace Similarity

I. INTRODUCTION

Healthcare processes can be subdivided into two groups: medical diagnosis/treatment processes and organizational/administrative processes [1]. In this paper we focus solely on the former. These kinds of processes are characterized as dynamic, multi-disciplinary, flexible, human-centric and knowledge-intensive processes [2], [3]. Consequently, the process actors executing the process (i.e., doctors and other medical personnel) are constantly confronted with decisions on what to do next from a large and diverse set of options. Decision making can be very demanding and even overwhelming, certainly for less experienced process actors. The current IT infrastructure remains lacking in support for decision making [4]. This is one of the reasons for requiring internships for new personnel. It gives them time to get a feel of how things are done in a certain department by following around more experienced personnel. However, these intern periods are time and effort consuming, not to mention expensive. Adding an additional channel for externalizing the knowledge of more experienced process actors as guidance, without them having to do anything extra, would have clear benefits [5]. Not only for less experienced process actors, but also for the other process actors. The result will be that the offered service will be more uniform, intern periods can be shortened, and process actors get relieved of some of the daily pressure.

In this paper, we take the first step towards offering such guidance in the form of recommendations for flexible and knowledge-intensive processes, and more specifically healthcare processes concerning the medical diagnosis and treatment of patients. The purpose of these recommendations is not to steer the process according to some optimization criteria, but rather to offer a direct reflection of the experience encapsulated in previous executions of the process. The results are returned to the users as process recommendations: a ranked list of the activities that are predicted to potentially be executed as the next activity and the corresponding probability estimations based on the most similar previously completed executions. The performance of a variety of strategies was measured and compared. The strategies can differ in what information they focus on, but they all leverage similarities between the currently executing process instance and the available historic data. The research questions (RQ) of this paper are the following:

1. Which strategies perform well for knowledge-intensive and flexible healthcare processes?
2. What kind of precision and consistency can be achieved and in what amount of computation time?
3. How much data is needed to for near-optimal precision?
4. Which strategies could be useful benchmarks to evaluate more advanced techniques (e.g., neural networks) in the future?

The strategies were implemented and applied to three different healthcare data sets: two real-life event logs and one artificial log. They are evaluated on average accuracy, multi-class brier score, log loss, how consistent the recommendation rankings are, and the required computation time.

The remainder of this paper is structured as follows. Section II gives an overview of the related research. The relevant terminology and formal problem definition are presented in section III. In section IV, the strategies used to generate recommendations are explained. Section V describes the experiments and discusses the corresponding results. And
Finally, the paper is concluded and directions for future research are suggested in section VI.

II. RELATED RESEARCH

The domain of process recommendations as a way to offer guidance to the process actors is still in its infancy. The prediction, ranking and probability estimation of potential next activities as a direct reflection of the experience encapsulated in previous executions of the process is, to the best of our knowledge, yet to be tackled. However, process recommendations have been proposed with other goals in mind. [6] uses three different trace abstractions to generate recommendations based on historic executions that minimize the cycle time of a partially executed process instance. The trace abstractions (prefix, set and multi-set) have also been used in this paper. [7] proposes a recommendation system for process-aware information systems that generates an enactment plan for the remainder of a partially executed process instance using constraint-based approach with a declarative process model as input. The differences with this paper are that only one enactment plan is proposed at any one time, so no more than one recommended next activity per available resource (i.e., no ranking), and an enactment plan specifies the remainder of the process instance (i.e., all recommended remaining activities in a specified order). The enactment plans are also optimized according the performance goals of the process, whereas the goal of this paper is to reflect the current way of working.

The related domain of predictive process monitoring is concerned with the continuous generation of predictions (often referred to as next-element predictions) of what activities should be performed and what input data values to provide, so that the likelihood of violation of business constraints is minimized [8], [9]. A business constraint can be a business goal, a desired outcome or a range for a key performance indicator (KPI) (e.g., instance duration should be less than 24 hours). The prediction of the next, the last or all remaining activities as well as the similarity to previously executed instances is used to estimate the likelihood of compliance based on the current state of the instance and the available historic data. The key difference to this paper is that the goal of predictive process monitoring is typically to indicate when special attention, or even an intervention, is needed to prevent the process instances from reaching an undesirable state. So, the prediction of the likelihood of compliance to some predicated constraint is of primary concern, more so than the precision of the next-activity prediction. Therefore, the prediction of the next activity is typically just a means to an end and not necessarily conveyed to the process actors. In this paper, our focus is solely on the prediction, ranking and probability estimation of potential next activities as a direct reflection of the experience encapsulated in previous executions of the process and we make no attempt to optimize the likelihood of achieving some business goal or to keep some KPI within an optimal range. In real-life applications, the ranking and probability estimations of the potential next activities proposed in this paper would be made available to process actors as a form of guidance. As a result, the primary focus is on the precision of the predictions as well as the relative ranking and the robustness of the ranking. Some predictive process monitoring systems can also provide recommendations as a way of intervening when potential goal violations loom.

This is an attempt to sway the process actor to follow historic executions that do satisfy the predicated constraint, as opposed to other historic executions that do not. The difference here is that they value certain similar historic executions more than others based on their outcomes, as opposed to just the direct similarity between the current execution and historic executions as is the goal of this paper.

Several techniques used in predictive process monitoring could be useful in the context of this paper, but it is difficult to compare the results from those papers with ours as the evaluation setup and criteria are different and the implementation, if available, often needs some adjustments to align with the goals of this paper. [10] apply a clustering technique and a classifier to estimate whether a given predicate will be satisfied upon completion of a currently executing process instance. [11] use recurrent neural networks with long short-term memory to predict the remaining time to completion and the next activity of partially executed process instances and [12] further investigated its applicability. [13] uses process mining techniques to predict the next activity and a classifier for to predict future cost and duration of a partially executed process instance.

Another related domain is that of trace clustering. Trace clustering techniques are often applied to create clusters of similar completed process executions before process mining techniques are applied to create a model of each cluster. These techniques should not be applied to cluster a partially executed instance in clusters of completed executions, as this is a completely different problem. However, some techniques use traces abstractions as a first step towards comparing the similarity of traces and the and we have implemented several of these as strategies in this paper. From [14] we used the activity and transition trace profiles, from [15] we used the control-flow manipulations CFP1 and CFP3. Additionally, many cluster distance measures can be also be used as similarity metrics. We started from the distance measures in [16] (e.g., Euclidean, Jaccard...) and added several others from different sources. The resulting strategies in a way make just one cluster with the abstraction of the partially executed instance as center. To calculate cluster distance (i.e., the similarity) between a partial and a completed instance, we introduced the concept of a variable-position similarity scorer.

III. PROBLEM STATEMENT

First, some definitions of terms or abbreviations that will be used throughout the paper:

- **Activity**: an identifiable action performed by a process actor as part of achieving a goal of the process. It can consume both time and resources and can produce data.
- **Activity event**: a timestamp that delineates a specific milestone (e.g., start or end) of the execution of an activity. The event itself does not consume any time or resource. IT-systems typically store these events, making it possible to reconstruct the chronology and durations of the actual activities of the process.
- **Data event**: a timestamp of when a specific data value was produced during the execution of a process. A data
event can also arise from external sources, so it does not necessarily have to be produced during an activity. IT-systems typically store these events, making it possible to reconstruct the chronology of the actual data produced during the execution of the process.

- **Trace**: a combination of chronologically ordered activity and data events that correspond to one execution of the processes (e.g., the diagnosis and treatment of a patient).

- **Prediction points in a trace**: a trace is divided into prediction points, each coming right after the completion of an activity and one just before the start of the first activity in the trace. For example, a trace <A, B> has three prediction points: before A, after A and after B. The time gaps between activities are not considered as separate prediction points.

- **Position in a trace**: each prediction point in a trace has a position. Consequently, the prediction point before the execution of the first activity is position 0, after the execution of the first activity is position 1, and so on...

- **Multiset**: a set in which elements are not necessarily unique (i.e., a list without a specified order).

- **HT (=historic trace)**: a trace of a previously completed process instance. The set of historic traces is used by the scorers to calculate recommendations for new partial traces. This set is also referred to as the learning set. Let H = [HT₀, ..., HTₖ] with k the number of traces in the learning set and HTᵢ = [Aᵢ₀, Aᵢ₁, ..., Aᵢₗ] with m the number of executed activities in trace HTᵢ.

- **HPT (=historic partial trace)**: a partial version of a HT where the activity and data events after a certain position in the HT are not considered.

- **CPT (=current partial trace)**: the partial trace for which the recommendations being are calculated. Let CPT = [Aᵢ₀, Aᵢ₀₋₁, ..., Aᵢ₋₁] with n the number of already performed activities.

- **NextActs**: the set of potential next activities for every prediction point. Let NextActs = {Aᵢ₀, ..., Aᵢₗ} with a the number of executable activities and Aᵢₗ representing the end of the trace (i.e., no more activities executed).

Now then we can provide a formal problem definition:

For each prediction point, predict \( A_{CPT(n+1)} = \{P₁, P₂, ..., Pₙ₊₁\} \) as a set of probability estimates for each potential next activity in NextActs given \( H \) so that \( A_{CPT(n+1)} \) matches the as-is process as close as possible. \( A_{CPT(n+1)} \) can subsequently be used to rank the activities of NextActs.

**IV. STRATEGIES**

We have identified 40 strategies to generate next-activity recommendations that all leverage trace similarity in one way or another. Most of these strategies have their roots in the related literature that came up when searching for topics like trace similarity, trace clustering, predictive process monitoring and vector/matrix similarity. We generalized, applied the same idea to other concepts and extrapolated some of those existing strategies to come up with additional strategies. And finally, we used our own creativity and domain knowledge to come up with the four more strategies (i.e., ActivityWithBefores, DataStateCustomOverlap and the combined strategies). We purposefully avoided more advanced techniques like neural networks, because of transparency concerns related to their black-box nature. However, in the future we will perform a comparison, including such advanced techniques, to evaluate if they can beat the strategies from this paper and whether the loss of transparency would be worth any potential added precision.

Each strategy was implemented in what we will refer to as a similarity scorer. These scorers take \( H \) and a CPT as input and return a list of potential next activities ranked according to their corresponding probability estimates. A scorer can therefore be regarded as the brains of an eventual recommendation system, with the other components being responsible for the integration of the recommendation system in the electronic health record system and for the user interface to convey the results to the users. The scorers are split into three general classes: pre-calculated, positionless and variable-position similarity scorers. Some of the scorers use certain weights and other parameter settings to function properly. The hyperparameter values used in the experiments are given below. Note that these values were just determined intuitively before running any experiments, so there is plenty of room for optimization through tuning.

**A. Pre-calculated similarity scorers**

These scorers calculate a frequency table once from \( H \) and use this table to generate a ranking of potential next activities and a corresponding probability estimate for the CPT. The computation times for these scorers presented in the experiments ignore the computation time of the frequency table, as this is a one-time cost (with a duration of just a couple of milliseconds).

- **AbsoluteFrequency** (=proportional guessing from [17]): calculates the absolute frequency in \( H \) of each activity in NextActs.
- **ActivityInTraceFrequency** (=set abstraction from [6] and similar to CFP2 from [15]): calculates the frequency of each activity in NextActs as the percentage of traces from \( H \) in which it occurs at least once.
- **RespondedFrequency** (=similar to CFP3 from [15]): calculates the frequency for each activity in NextActs, and also for the start of trace, that it is followed by a certain other activity from NextActs according to the traces in \( H \). The size of the frequency table increases quadratically with the number of activities.
- **StepFrequency** (=Prefix abstraction from [6]): calculates the frequency of each activity in NextActs for each possible position as used in \( H \).

**B. Positionless similarity scorers**

These scorers calculate the similarity at runtime between the CPT and the completed traces in \( H \), so they do not consider any partial versions of the traces in \( H \).

- **IntraTraceFrequency** (=multi-set abstraction from [6]): calculates a percentage for each activity in NextActs as the number of traces in \( H \) that have more occurrences of the activity than the CPT, divided the sum of this number for all activities in NextActs.
IntraTraceFrequencyNotNull: the same as the IntraTraceFrequency scorer, but without considering the end of the trace as a separate activity. The thought process was that adding this extra activity, which occurs eventually in every HT, might be detrimental to the performance of the IntraTraceFrequency scorer because the added activity will potentially result in a higher rank and a lower probability estimate of the recommendation of the correct next activity for each prediction point except for the last one of a trace.

C. Variable-position similarity scorers

These scorers calculate the similarity of the CPT with a HT by calculating the similarity of the CPT with all possible HPTs of a HT that stop in different positions. The similarity is not necessarily calculated for all possible HPTs of a HT, but rather heuristically. The current position in the CPT (i.e., the number of activities already executed for the CPT) is used as a starting point, and various positions left and right of the start position are considered until the stop criteria is reached for each direction. This criteria states that the search is stopped is there is no local improvement in the scores of 4 consecutive HPTs in that direction. After calculating the similarity score for each trace in H, the 100 most similar HTs are kept for calculating the probability estimates of the potential next activities. This ‘100’ and ‘4’ are parameters that can be further tuned in the future.

UniqueActivity: calculates the similarity as the number of unique activities that were executed in both (partial) traces divided by the total number of unique activities executed in the CPT, minus a penalty of 0.001 times the number of unique activities that differ between both traces.

Activity (activity trace profile from [14] and CFP1 from [15]): calculates the similarity as the weighted sum of two sub(scores, minus two penalty factors. The first subscore (weight = 0.7857) is the number of unique activities that were executed in both traces divided by the total number of unique activities executed in the CPT. The second subscore (weight = 0.2143) is the complement of the sum of the absolute differences between the number of repeated executions of shared activities of both traces, divided by the total number of activities executed in the CPT. The penalty factors are 0.001 times the number of unique activities not shared among both traces and 0.0001 times the number of repeated activity transitions that are not shared among both traces.

ActivityTransition (=transition trace profile from [14]): calculates the similarity as the weighted sum of two sub(scores, minus two penalty factors. The first subscore (weight = 0.7857) is the number of unique activity transitions that are part of both traces, divided by the total number of unique activity transitions in the CPT. The second subscore (weight = 0.2143) is the complement of the sum of the absolute differences between the number of repetitions of activity transitions that are part of both traces, divided by the total number of activity transitions in the CPT. The penalty factors are 0.001 times the number of unique activity transitions not shared among both traces and 0.0001 times the number of repeated activity transitions that are not shared among both traces.

DataStateCustomOverlap: calculates the similarity as the number of shared currently active data attributes in both traces, divided by the total number of currently active data attributes in the CPT, minus a penalty of 0.01 times the number of unique activities that differ between both traces.

We also considered the following general mathematical distance and similarity metrics (Euclidian and Jaccard from [16], Tanimoto from [18] and the others were taken from the SimMetrics Java library1 on the condition that they could be easily applied in this context):

- **Dice**: defined as twice the shared information (intersection) of two sets divided by sum of cardinalities.
- **Jaccard**: defined as the size of the intersection of two multisets divided by the size of the union of the sets.
- **Overlap Coefficient**: defined as the size of the intersection of two multisets divided by the size of the set that originates from the CPT.
- **Tanimoto Coefficient**: defined as the cosine of the angle between two sets expressed as sparse vectors.
- **Block Distance**: defined as the sum of the absolute differences of the Cartesian coordinates of two multisets. Also known as taxicab geometry, rectilinear distance, L1 distance, snake distance, city block distance, Manhattan distance or Manhattan length.
- **Euclidean Distance**: defined as the "ordinary" straight-line distance between two multisets in Euclidean space.
- **Simon White**: Dice, but with multisets.
- **Generalized Jaccard**: Jaccard, but with multisets.
- **Generalized Overlap Coefficient**: same as Overlap Coefficient, but with multisets.
- **Cosine Similarity**: Tanimoto Coefficient, but with multisets.

For each of these similarity metrics we defined two scorers, and an additional third for all but the Simon White, Generalized Jaccard, Generalized Overlap Coefficient and Cosine Similarity

1. https://github.com/Simmetrics/simmetrics
metrics (as these would coincide with the non-multiset versions of the corresponding metric scorers):

- **Activity...Scorer**: the lists of activities events of the CPT and HPT are converted to sets or multisets, respectively, to calculate the metric.
- **Data...Scorer**: the lists of data events of the CPT and HPT are converted to sets or multisets, respectively, to calculate the metric.
- **DataState...Scorer**: the data states of the CPT and HPT are converted to sets or multisets, respectively, to calculate the metric. The data state is the set of data attributes that are active at the current point in the trace.

**D. Combined scorers**

The scorers above all use an abstraction of just one aspect of the trace. Of course, the eventual goal is to create a scorer that takes multiple aspects into account. As an example of this, we also implemented two scorers that combine several of the aforementioned scorers.

**ActivityWithBeforesAndData**: calculates the similarity as the weighted sum of two subscores. The first subscore is the ActivityWithBefores score (weight = 0.6) and the other is the DataStateCustomOverlap score (weight = 0.4).

**ActivityWithBeforesAndDataAndKBs**: calculates a score for each potential next activity as a weighted combination of 5 scorers: ActivityWithBefores, DataStateCustomOverlap, RespondedFrequency, StepFrequency and AbsoluteFrequency. The first two are used to calculate a similarity score just like the ActivityWithBeforesAndData scorer, but with slightly different weights (weight of ActivityWithBefores = 0.4/0.59 and weight of DataStateCustomOverlap = 0.19/0.59). The final score is then calculated as the sum of ActivityWithBeforesAndData score (weight = 0.59), the RespondedFrequency score (weight = 0.4), the StepFrequency score (weight = 0.009) and the AbsoluteFrequency score (weight = 0.001).

**V. EXPERIMENTS**

We performed two experiments to compare the performance of the different strategies. The first experiment tries to answer RQ1, RQ2 and RQ4 by applying the scorers to three healthcare data sets and comparing the performance of the strategies on different types of data sets. The second experiment is similar, but it varies the size of H to investigate the relation between the predictive power and the size of H to answer RQ3.

**A. Experiment 1: comparing the general predictive power over different data sets**

1) **Experimental setup**

Three healthcare event logs were used:

- **ArmFractures**: an artificial event log containing 5000 traces (31980 activity and 45147 data events using 9 unique activities and 21 unique data values) about patients with suspected arm fractures. It spans parts of the hospital emergency and orthopedic departments and was created via interviews with domain experts.
- **Sepsis**: a real-life event log containing 1050 traces (15214 activity and 32885 data events using 16 unique activities and 214 unique data values) specifically about sepsis cases that were extracted from the Enterprise Resource Planning system of a hospital.
- **EmergencyDepartment**: a real-life event log containing 41657 traces (625758 activity and 1350465 data events using 116 unique activities and 951 unique data values) about the registration, triage, diagnosis and treatment of patients in the medical emergency department of a Belgian hospital. We are not at liberty to make this event log public due to a confidentiality agreement.

These event logs contain activity events as well as the relevant data events with the corresponding data values entered by the medical personnel during the execution of the process. The logs can all be categorized as flexible and knowledge-intensive processes, but they do differ in how extreme they exhibit these properties. The EmergencyDepartment-log is the most complex data set, yet, it also has the most traces and details for the scores to base their recommendations on.

In this experiment, each scorer was used to calculate recommendations for each prediction point of each trace from the log with all other traces from that log as H. The experiments ran on an Intel Xeon Gold 6140 with 32GB of RAM. The 36-cores were used to calculate the similarity scores of up to 36 traces from H in parallel. The results are evaluated using the trace-wise averages of four performance measures, as opposed to the overall averages. Both averages will give similar results, but the overall averages have the undesired effect of giving longer traces a relatively higher weight because they have more prediction points. The longer traces are typically also harder to predict. Hence, they would skew the results. Trace-wise averages are more representational for general performance.

The three following well-known classification performance measures used to evaluate the performance of the scorers: **accuracy, log loss and multi-class brier score** [19]. These measures quantify the precision of the recommendations (RQ1 and RQ2) by comparing, respectively, the highest ranked recommendation, probability estimate of the highest ranked recommendation and probability estimates of each recommendation to the actual next activity. This gives us a lot of information on the relative predictive power of each strategy. However, the purpose of recommendations in this paper is not merely to predict, but also as a tool to offer guidance to process actors. Therefore, we opted to define an additional performance measure that scores the consistency of the ranking of recommendations (RQ2): the **rank score**. Where the classification performance measures just make a distinction between right and wrong (e.g., ranked first), the rank score makes a distinction between levels of ‘wrong’. For example,
ranking the correct next activity as second or 15th makes a big practical difference as process actors cannot be expected to consider much more than a few recommendations at the top.

The rank score is calculated as follows:

$$\text{Rank score} = \left( \sum_{i=1}^{c} f_i \cdot r_i \right)^2$$

With $c$ denoting the number of prediction outcomes/classes (i.e., the size of NextActs), $f_i$ representing whether activity $i$ was the correct next activity (i.e., $f_i \in \{0, 1\}$ and $f_i = 1$ iff activity $i$ was the correct next activity) and $r_i$ representing the ranking of potential next activity $i$.

The ideal rank score is equal to one, as this means that the correct next activity was ranked as first. Sub-optimal rankings will result in a rank score of more than one. The formula is squared so that worse rankings are exponentially penalized. This puts strategies that consistently have the correct next activity at the top of the ranking at an advantage.

2) Computational results

The results of experiment 1 are summarized in Table I (these results and some additional details are available at https://github.com/stevemert/recommendationExperiments). The average rank of the correct next activity is also included, as this is a more relatable than the quite abstract performance measures.

By design, the pre-calculated scorers are easily the fastest. But it is noteworthy that even the slowest scorer only needed an average of little more than 0.6 seconds of computation time for even the biggest and most complex event log (RQ2).

The color-coding of the table reveals that the results for the different performance measures are clearly correlated. A relatively good result for one measure usually also indicates relatively good performance for the others. In general, the variable-position activity similarity scorers come out on top. Their performance was consistently among the best for all logs. The differences between the different activity similarity scorers are small, but the Activity and ActivityWithBefores scorers consistently perform just slightly better (RQ1). The data and data state variable-position similarity scorers perform poorly across the board. This is not a big surprise, as predicting next activities without knowing what activities have already been performed is inherently more difficult. The gap is smaller for the data state scorers on the data-heavy EmergencyDepartment log. The performance of the positionless similarity scorers is below average. The distinction between the IntraTraceFrequency and IntraTraceFrequencyNotNull scorers was made due to concerns about the impact of adding the end of trace prediction. The results show that these concerns were legitimate. The IntraTraceFrequencyNotNull scorer has a higher accuracy for all logs compared to the IntraTraceFrequency scorer, however, the trade-off is a much higher log loss and rank score.

The performance of the pre-calculated scorers is a bit of a mixed bag. The AbsoluteFrequency and ActivityInTraceFrequency scorers perform poorly, the StepFrequency scorer performs average and the performance of the RespondedFrequency scorer is not that far of the top performers. The latter even generally outperforms the activity similarity scorers on the EmergencyDepartment log. They have similar accuracy and brier scores, but RespondedFrequency scorer has a significantly better log loss and rank score. A closer analysis points the finger to the configuration of the similarity algorithm. We only used the 100 most similar traces from $H$ to calculate the probability estimates of each potential next activity. Therefore, not all potential next activities are ranked for logs with close-to-or-more activities. If a scorer did not rank the correct next activity, we calculated its rank as the average ranking of all unranked recommendations. For example, the EmergencyDepartment log has 117 potential next activities (=116 activities + 1 end of trace). Suppose a scorer ranked 25 potential next activities, but the correct next activity was not among them. Then the rank of the correct next activity will be calculated as $25 + (117-25)/2 = 71$. Increasing the number of similar traces to consider could improve this, however, this would result in an increase in computation time.

The use of a data state similarity scorer to complement an activity similarity in the ActivityWithBeforesAndData scorer had mixed results. The combined scorer shows a slight improvement in the ArmFractures log compared to just the ActivityWithBefores scorer, but the results are opposite for the other two logs. We suspect that the chosen weights might have been the issue. The weight of the DataStateCustomOverlap similarity score is perhaps a too high (40%). This component has a slightly lower weight (32%), compared to its ActivityWithBefores component, in the slightly better performing ActivityWithBeforesAndDataAndKBs scorer. However, the increase in performance could also come from its RespondedFrequency component. Although, further experiments will be needed to draw any real conclusions about the complementary value of a data similarity scorer, the ActivityWithBeforesAndDataAndKBs scorer clearly combines the best of both worlds (RQ1). It has the predictive power of the similarity scorers and the consistency of the pre-calculated scorers for logs with a high number of activities. And finally, it is also worth noting that the computation time of the combined scorers typically is lower than the sum of the computation times of its component scorers. The explanation can be found in how the component similarity scorers are combined. They do not run in isolation, instead the combination of the similarity components is used directly in the variable-position algorithm. Thus, the convergence of this heuristic can differ from the convergence of the component scorers in isolation, albeit using an inherently slower similarity calculation.

B. Experiment 2: the relation between predictive power and the size of $H$

1) Experimental setup

In this experiment, we created sets of varying size to serve as $H$ and sets of traces to evaluate the scorers with. The two sets were randomly created and never contained overlapping traces. For each log, the size of the set of traces for evaluation was always the same for all sizes of $H$ (500, 100 and 1657, respectively). The experiment ran 3 to 5 times for each size and the average over all runs was calculated. Otherwise, the setup and evaluation criteria were the same as in experiment 1.

2) Computational results

The results of this experiment have been summarized in Fig. 1. The less interesting scorers (i.e., those that perform similar or
**TABLE I.** The results for experiment 1. For each column, the top performer is formatted in italics and bold and a color-coding was used to represent the best-to-worse ranking (green-yellow-red).
Fig. 1. The results for experiment 2
worse than comparable strategies in experiment 1) have been omitted to not overload the graphs (full versions and some additional details are available at https://github.com/stevmert/recommendationExperiments).

The trace-wise average of the accuracy, multi-class brier score, log loss and rank score of the similarity scorers all converge very quickly. The performance measures still improve with more and more traces to learn from, but the added value gets smaller and smaller. This corresponds to a logarithmic relation: with a high base number for the accuracy and a fractional base number for the other performance measures. For example, the ActivityWithBeforeAndDataAndKBs scorer already reaches an average accuracy of 57.75% with 500 traces in H, and this increases by just 2.52% with 39500 additional traces in H. This is a small improvement in accuracy compared to the increase in computation time by more than a factor 40 (RQ3).

C. Discussion

The results of the two experiments demonstrate the predictive power of the the of the considered strategies for generating ranked busi process recommendation lists as a direct reflection of the experience encapsulated in previous executions of the process. We can formulate some conclusions for each research question.

RQ1: Which strategies perform well for knowledge-intensive and flexible healthcare processes?

- For logs with a low number of activities, the variable-position activity similarity scorers perform significantly better than the pre-calculated, positionless and variable-position data similarity scorers. Therefore, they show a lot of promise as a high-weight component for combined strategies.
- The performance of the variable-position similarity scorers clearly declines for logs with a high number of activities, because of the limited number of ranked next activities. In this case, the penalty of a correct next activity being unranked weighs heavily on the performance. As we only had one data set with a high number of activities, it is hard to generalize this further or to even use a stricter formulation. The lower performance could also be attributed to the high complexity of the real-life EmergencyDepartement. Future experiments will determine if increasing the number of most similar traces to consider is a good countermeasure and what would be the corresponding impact on the computation times.
- The data and data state variable-position similarity scorers perform unsatisfactory across the board. This was expected, as we envisioned them to be more of an auxiliary and low-weight component for combined scorers. The data state strategies show the most potential.
- Overall, the combined scorers come out on top for all performance measures. Only for the Sepsis log they are beaten by a few variable-position activity similarity scorers on the brier and rank scores, but we suspect this might be due to suboptimal weights and a limited number of traces to learn from (i.e., 1050 compared to 5000 and 41657 for the other event logs). Choosing the subcomponents and weights more carefully, keeping in mind the experimental results from this paper, could probably also get them to the top for the Sepsis log. Hyperparameter tuning algorithms can be applied to do this automatically.

The Sepsis log was also used in [17], which compares several other next-element prediction strategies. As explained in section II, the goal of these predictions is a bit different to that of this paper. However, the raw predictive power of the techniques can still be compared. Tax et al. use only the brier score to compare the performance of the techniques. Our best strategies outperform the baseline, process mining, machine learning compression, Markov and grammar inference techniques considered in that paper. The artificial neural network and all k-order Markov models (AKON) techniques did have better brier scores, but difference is small (0.0004 and 0.0019, respectively). Perhaps a more meticulous selection of subcomponents for a new combined scorer and some form of hyperparameter tuning would result in similar or even better brier scores. It is worth noting that a better brier score, although they are clearly correlated, does not necessarily mean that the other performance measures used in this paper would agree. For example, the Activity scorer has a better average brier score for the Sepsis log in the first experiment, but a worse average rank, accuracy, log loss and rank score. Comparing the artificial neural network and all k-order Markov models (AKON) techniques using all of our performance measures on a more complex data set like the EmergencyDepartment log will hopefully provide a clearer picture in the future.

RQ2: What kind of precision and consistency can be achieved and in what amount of computation time?

- The achieved performance primarily depends on the characteristics of log. The variable-position activity similarity and combined scorers achieve an accuracy of almost 90% for the ArmFractures log, almost 64% for the Sepsis log and 60% for the EmergencyDepartment log. On average, the correct next activity will be in the top 2 of the variable-position activity similarity and combined scorers’ recommendations for the ArmFractures and Sepsis logs, in the top 4 and top 3 for the variable-position activity similarity and combined scorers, respectively, for the EmergencyDepartment log. The rank scores for the ArmFractures and Sepsis logs show that the variable-position activity similarity scorers perform consistently for logs with relatively few unique activities, but when that number rises they tend to struggle a bit more. The precalculated scorers achieve better performance for the EmergencyDepartment log with its 116 unique activities, while the variable-position combined scorer that combines both takes the crown in consistency.
- The strong performance of the variable-position activity similarity and combined scorers does come at the
expense of a higher average computation time. Where the pre-calculated scorers just need to retrieve the ranking from memory, the variable-position similarity scorers must calculate the similarity between the CPT and the traces in \( H \) on the spot and subsequently analyze the most similar traces in \( H \) for what activity each has as next at its most similar position. However, even the highest average computation time is considerably less than one second. This certainly seems to be acceptable for real-life applications. Moreover, the results of the second experiment show that a relatively small size for \( H \) can offer almost the same performance, while severely reducing the computation time.

- The pre-calculated scorers of course have the lowest computation time, but even the relatively slow scorers. This is an excellent result for these unoptimized implementations, albeit with more parallel processing power than might be typically available. It bodes well for its applicability in real-life settings, where a noticeably long computation time could be a source of frustration for the process actors using such a system.

RQ3: How much data is needed to for near-optimal precision?

- Experiment 2 demonstrates that just a couple of hundred traces in \( H \) is sufficient for most cases concerning the datasets from this paper or similar datasets. Large sizes for \( H \) are only sensible in situations where the maximal performance is required, and processing power is less of an issue. Of course, the exact number of traces required will strongly depend on the characteristics of process and the quality of the corresponding event log. Larger sizes of \( H \) could be needed to achieve near-optimal precision for processes with an even higher complexity or with datasets of lower quality.

RQ4: Which strategies could be useful benchmarks to evaluate more advanced techniques (e.g., neural networks) in the future?

- The RespondedFrequency scorer, and to a lesser degree also the StepFrequency scorer, manages to consistently offer moderate-to-good performance, independent of the complexity and number of activities. The simplicity of the strategy and negligible computation time make it an easy-to-implement (or even fallback) option to provide some guidance. It can also be an excellent benchmark for more advanced strategies and techniques. The performance of the AbsoluteFrequency, ActivityInTraceFrequency, IntraTraceFrequency and IntraTraceFrequencyNotNull scorers is insufficient for these purposes. Their use should be limited to providing relative rankings for otherwise unranked or extremely unlikely next activities, as was the case for the ActivityWithBeforesAndDataAndKBs scorer.

VI. CONCLUSION AND FUTURE RESEARCH

The results of the two experiments demonstrate the predictive power of the of the considered strategies when generating ranked process recommendation lists as a direct reflection of the experience encapsulated in previous executions of the process. Each strategy focuses on specific aspects of the process and they use all sorts of similarity metrics to relate partially executed traces to historic executions.

Although the results vary from log to log and even from performance measure to performance measure, some straightforward trends were revealed. Firstly, there are huge performance differences between the set of simple and pre-calculated strategies. The RespondedFrequency scorer is the clear-cut winner among this group of strategies with its consistent and moderate-to-good overall performance. This scorer seems like a great choice for a performance benchmark for more advanced techniques. Secondly, the variable-position activity similarity scorers unsurprisingly are top of the class of the scorers that focus on just one aspect of the process. Even for knowledge-intensive processes, previously executed activities seem like the best single-aspect predictors for the future activities, albeit with some meaningful differences between different strategies to represent the previously executed activities. Thirdly, the data and data state variable-position state similarity scorers can be useful as a component that complements another strategy, but not as a standalone scorer, and from this group of strategies, the data strategies that keep a data state are superior to those that do not. Fourthly, the drawbacks of single-aspect strategies can be mitigated by combining strategies. The results of the combined scorers demonstrate that combinations of strategies can outperform the separate building blocks, at a cost of a relatively small increase in computation time. Fifthly, the precision from the simple and unoptimized scorers used in this paper is promising. For the ArmFractures log they could predict the correct next activity with an accuracy of just a hair under 90%. The two real-life logs were clearly more challenging as reflected by a maximal accuracy of 64% and 60%. However, this just reflects the number of times that the correct next activity was predicted as the first ranked recommendation. For offering guidance to the process actors it is still acceptable if the correct next activity was ranked near the top. The best performing scorer were able to rank the correct next activity in the top-2 or top-3 of their recommendations on average. This should ensure that the correct recommendation was at least within reasonable viewing range of the process actors. And finally, the computation time for the different strategies is manageable, even for complex real-live event logs. If necessary, the computation time can be reduced significantly by reducing the size of \( H \) (up to a small fraction of all available data) with relatively minimal impact on the overall performance of the scorer.

As part of the future research, we plan to investigate the impact of increasing the number of similar traces used in the variable-position similarity scorers and the use of trace encodings to improve memory usage. For the scorers themselves, we will consider optimizations, hyper parameter tuning and more combinations of strategies. More advanced techniques like neural networks will also be considered and compared. On a more general level, we would like to apply the techniques to many more event logs. The idea is that this might reveal preferred strategies or techniques for processes on different parts of the flexibility and/or knowledge-intensity spectrum.
REFERENCES


