Genetic Correlation Discovery for Unlabeled Event Logs

Pascal Schulze and Anjo Seidel

Hasso Plattner Institute
University of Potsdam, Germany
{pascal.schulze,anjo.seidel}@uni-potsdam.de

Abstract. tbd

Keywords: process mining · process discovery · genetic mining · event logs · workflow patterns

1 Introduction

The economical urge ....

Most of the used and proven mining techniques, conformance checks, and performance analyzes out there require event logs where at least case ID, activity ID and the timestamp are existent. Should one of these attribute be missing on the event log these techniques cannot be applied to this input.

- Process Mining
- Event Log Generation
- Event Correlation
- Unlabeled Event Logs
- Illustrate need of finding case IDs
- mention "Process Discovery of unlabeled event logs", strong assumptions
- lack of research and potential of improvements
- Approach: Genetic Programming Paradigm

The described implementation in this paper is free to use under the MIT license and publicly accessible at `www.github.com/pscls/genetic-process-discovery`[1]

2 Related Work

There have already been developed several techniques to facilitate event correlation on unlabeled event logs. These thought out approaches deduce the missing case identifier for every event by using different concepts and often rely on the existence of knowledge.

In 2015 Pourmirza et al. [5] published there idea of a proverbial correlation miner. The concept behind there miner is a merging, clustering, and aggregation

[1] The code builds and extends the work of Ferreira et al. [2]

process on the different cases in the event log to eventually generate a suitable orchestration model based on assigned identifiers. However, there miner requires a timestamp to every event to be applicable. As well in 2015 Helal et al. [4] and in 2016 Bayomie et al [1] propose techniques which both depend on the executed process model and heuristic information of the different activities within the process as input in addition to the execution log. The idea behind these is to compensate the unlabeled characteristic with further knowledge to finally come up the an derived labeled event log.

All this set beside, Ferreira et al. [3] already came up with a zero-knowledge approach in 2009 which uses a probabilistic and iterative expectation–maximization procedure to estimate the case identifiers. Thereby, they only require the identifiers and chronology of the events - an unquestioned unlabeled event log.

To the best of our knowledge, we cannot provide any related work to the approach presented in this paper to solve the correlation problem by using a genetic algorithm.

3 Background

In this Section we will give an brief insight into the approach presented[2] by Ferreira's et al. [3], which is the core of our genetic extension. Furthermore, we will define the test environment as well as the accuracy which is going to be used for algorithm accuracy comparisons.

3.1 Genetic Programming Paradigm

The concept of genetic programming comes from the nature itself. It is an evolutionary approach where the program will explore the possible solution space on its own with the help of a defined quality or fitness function and then evolve a set of solution candidates, instead of only training one, by mimicking the basic principles of Darwinian evolution [6]. After each training epoch/generation we measure the fitness of all individuals and rank them directly or let them 'fight' against each other in a tournament - known as *selection*. The fittest will be chosen to *reproduce* the next generation of offsprings which will naturally replace the weakest ones. The characteristics and features of the best individuals will be transferred to the next generation while weak properties will be wiped out. As in nature *mutations* are happening unpredictable and will change random properties slightly in the next generation to keep avoid getting stuck at one place.

3.2 Greedy Case ID Estimation

The present approach by Ferreira's et al. [3] takes as input a symbol sequence, which consists of event labels. These labels stand for a specific event type and

[2] We will often refer to this as the *base algorithm*.

provide no further information than that. The aim of the algorithm is to correlate those events to traces, where each trace is a sequence of events that belong to the same case ID.

For this correlation, a probabilistic method is presented. The correlation is derived from a probability matrix, which holds the probabilities for each sequence of two following events. This probability matrix is initialized as the so called M+ matirx. This M+ is a probability matrix, which derives its probabilities from the input symbol sequence. The probability of two subsequent events is computed by the number of times that the later event is a direct successor in the input symbol sequence. With that probability matrix, traces from the input symbol sequence can now be estimated. Here, we use a greedy algorithm. It creates a new trace for each start event. For all other events it assigns this event to the one existing trace, which last event has the highest probability of being predecessor of the new event. In the end, now we have estimated traces, which are subject to the initial probability matrix. In order to reduce the subjectivity of the initial matrix, the probability matrix can be computed from the event successorships in the estimated traces. The new probability matrix now holds more precise probabilities than M+. With that new matrix, the traces can be estimated again from the input symbol sequence. The procedure of estimating traces based on a matrix and computing the matrix from the previously estimated traces can be run iteratively until the matrix does not change anymore and is stable. At that point, the last estimated traces are the output of this algorithm. Each of those estimated traces represents one case and a case ID can be assigned.

3.3 Base Accuracy

Do we wanna wright something like this about hor found mismatch?

The described accuracy for the base algorithm by Ferreira et al. converges against an average G-score of 0.6 “for input event logs with 300 sources and varying number of overlapping sources [while] each point has been obtained by averaging over 1000 synthetically-generated logs” [3]. However, during the research we noted that combining this setting with the provided python paper-code [2] results in far better G-score with an asymptote of around 0.82 as it can be seen in figure 2. On the basis of this found discrepancy we will use our observed accuracy whenever comparing our extension with the base algorithm from this point on.

4 The Genetic Approach

The weakness of Ferreira’s et al. algorithm lays in the single and iterative training model. Every model is a completely self-contained and autonomous unit which relies strongly on a good initialization. The presented initialization function M^+ provides good to great results in general. But, should the first guess be significantly off, the model will probably never improve its prediction to a usable state based on our observations. Thus in a real world scenario where we

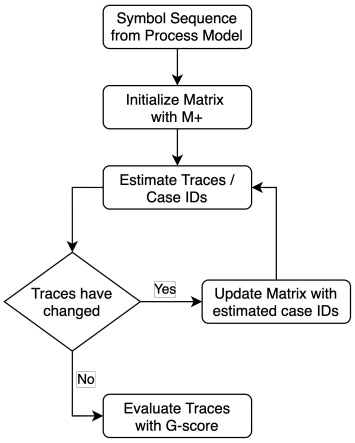

Fig. 1. Training's flow of the base algorithm [3]

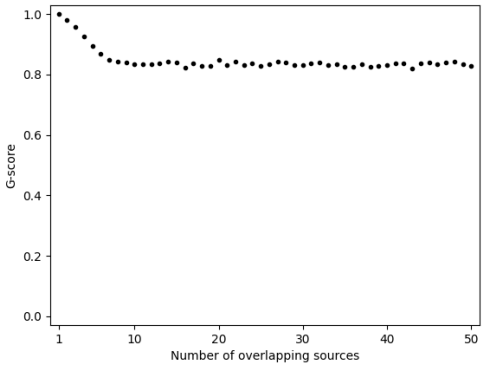

Fig. 2. Token Replay on Models

assume a defined process model is not existing we have to trust the output without any chance to question the accuracy of it. By extending the core algorithm with a genetic concept we can train multiple models simultaneously and connect them to slowly find an (local) optimum in a far wider field of range to overcome possible bad initialization by building on the fittest models in each epoch.

According to the genetic programming paradigm described in **??** an extension for the base algorithm was provided. This extension joins the concept of a genetic algorithm and the iterative expectation maximization algorithm. The result is described in the following and can be seen in 3.

As previously described, a genetic algorithm runs on multiple instances. In our case, the multiple probability matrices are estimated from the input symbol sequence x. Those n matrices are generated analog to the base algorithm as M+ matrices. In order to force variance within the models and to possibly produce better matrices than M+, the matrices are mutated randomly. Every entry of every matrix is mutated with a certain probability and if so with a certain range of factors.

Thus n slightly different probability matrices are generated. Analog to the base algorithm, for each of the matrices $M \epsilon M'$ and the symbol sequence x, the traces s can be estimated. Each model instance m now consists of the input sequence x, the probability matrix M and estimated sequences s. Again, like in the base algorithm, we can then compute a new M from x and s for all model instances. Those updated matrices will then be used in the selection step of the genetic algorithm.

With the help of a fitness function, which sorts all models according to their fitness to the process, the best ranked models can be reproduced. For the reproduction, the best ranked matrices will produce new offsprings. Two matrices can be merged by respectively building the average of all their values. That way,

probability matrices are generated that possibly represent the underlying process better than the weaker matrices. Complementary, the lowest ranked and therefore weakest matrices can be excluded from the genetic algorithm.
This new set of probability matrices can now again be randomly mutated. This provokes the possible creation of even fitter models.
With an improved set of matrices, a new generation of traces s can be estimated based on the new probability matrices M' and the input sequence s.
The process of estimating traces, deriving new probability matrices that are ranked, selected, reproduced and mutated, only to be the base of a new trace estimation, is run repetitively for a certain number of epochs. If this number of epochs is reached, the best ranked and therefore fittest matrix from the last model evaluation is returned as the output.
This output matrix is the result of the genetic expectation maximization algorithm and can be further used for solving the correlation problem of unlabelled event logs or for evaluation of the approach.

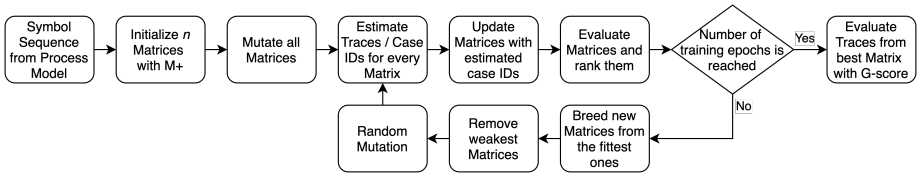

Fig. 3. Training's flow with the genetic extension.

The implementation of the the described algorithm is quite uncomplicated except for one crucial step. The used fitness function.....

5 Fitness Function

As in many other use cases, mapping functions from any kind of input to a specific number range are indispensable to evaluate and thereby rank the different input entries to measure the accuracy. For our usage such a function, which takes a model as input and assigns a value to it, already exists and is described by Ferreira et al.[3] as the so called G-score:

$$G(m_1 \mid\mid m_2) \widehat{=} \sum_{z \in Z} \sqrt{m_1(z) \cdot m_2(z)} \tag{1}$$

For every model m we will get the similarity to another model q - this could be the process model from which the events got distributed - by computing the geometric mean of both distributions. Thus, we get the true labeling accuracy of each predicted model. Unfortunately, we see that the G-score gets normally used

at the end to verify the produced traces and thereby takes the original underlying process model as one of the inputs. But as already described in chapter 4 we need to rank our models by evaluating each interim solution to the problem by measuring how "fit" they are during the prediction process where we assume we do not know to the process model due to the unlabeled event log problematic. As it is not straightforward to find a proper fitness function[3] for this use case, this section is about the requirements, concepts, characteristics, and results of such possible fitness functions or fitness approximation functions to assign a fitness value without having the ground truth.

5.1 Requirements

Every function which has a structure as described in equation 2 is a possible candidate of being a fitness or fitness-approximation for us since its result is a quantitatively indicator of how fit a given input m from the trainable model set M is. In our case a model m consists of a probability matrix[4] p and the estimated traces y to a given input event log l. The function needs to assign a model a higher value if the estimated traces and the probabilities of the matrix p represent the underlying process model better than other models.

$$f : m \mapsto \mathbb{R} \tag{2}$$

The Problem here is that we are working on an unlabeled event log - each model m is only one probabilistic prediction and thus does not contain any kind of quantitatable data. We must assume that every model instance is an equally likely candidate to have the highest fitness. Following, some kind of additional input information X is needed here. This X will be primarily M, the set of all model instances m' and their respective $p_{m'}$, $y_{m'}$, and $l_{m'}$. M holds information, that we can access without any restrictions during the genetic procedure.
It is also possible to give additional information into X, which implicates new assumptions to the algorithm. Additional event log(s) l' for the evaluation could be of use here. However, as the genetic approach is supposed to satisfy the same assumptions as presented by Fereiera et al., we will use M.

$$f : m, M \mapsto \mathbb{R} \tag{3}$$

For one genetic run of e epochs with n models the fitness function gets called $e \cdot n$ times[5]. Therefore the a sufficiently fast function is necessary to minimize the possibility of being the computational bottleneck.

[3] Also referred to as weight or evaluation function in our context.
[4] A different representation of a Markov chain.
[5] For the whole evaluation process we need to execute 1000 experiments for each overlapping from 1 to 50 with arbitrarily set hyperparameters e and n which results in calling the fitness function $50 \cdot 1000 \cdot e \cdot n$ times.

5.2 Prospect Functions

The limited information that is available as input is complicating the provision of a sufficient fitness function. THE GENERAL IDEA... In the following we will present the ideas and their performance in manners of sorting the models $m' \epsilon M$

Alignment and Token Replay Alignment and Token Replay have been proven to be powerful techniques in the context of conformance checking. As the fitness function f aims to rank models $m \epsilon M$ according to the conformance among each other, the use of those techniques seems to be obvious.
Alignment... (Definition...). However, it has been shown that the alignment of multiple traces at once is an NP-complete problem. Therefore, Alignment seems to be unpromising as a fitness function in matters of run-time and complexity.
Token Replay, on the other hand, can be managed for multiple traces (Source?). Originally, Token Replay calculates the fitness of a model in a petri net or matrix representation ...Definition (Source?). Formula?. Again, as the fitness function has to evaluate one model m against all other models $m' \epsilon M$ and the transitions of m are probabilistic, the use of token replay is not trivial. We implemented different concepts to tackle this problem and evaluated the G-Score of the models in the accordingly sorted list. For evaluation of those fitness functions, the G-Score was calculated by comparing the models to the actual case IDs. The results are shown in **??**, **??** and **??**. For comparison 5 shows the distribution of G-Scores within the list with random sort. The results prove that these prospect fitness functions will not sort the models adequately for the genetic algorithm.

Overall Trace Probabilities The general idea of comparing one model against all other model instances is also fundamental for the next fitness function. The main idea is, again, to compare the matrix of our model instance m against the estimated traces of all other models.
The approach is based on some findings for the specific use case. First, the sum of the probabilities of all traces, which the matrix holds, is one. The reason for that is that the matrix is computed based on the estimated traces. (More Explanation) The best suiting set of traces for the matrix is just the set of traces of the same model instance, as the matrix was created based on those traces and the sum of probabilities of traces is one. Any other set of traces will resemble this matrix worse or at most equally good. The sum of their probabilities is always less or equal to one.
If we now take the general idea into consideration, a matrix with a high fitness represents all other sets of traces well. Therefore this matrix will have a high sum of trace probabilities for each other set of traces. And if so, the average of all summed up trace probabilities per set of traces is high in comparison to less fit matrices.
This results in the fitness function 4.

$$f(m, M) = \frac{1}{|M|} \cdot \sum_{m_i \in M} \left(\sum_{y \in Y_{m_i}} P_{m_i}(y) \right) \tag{4}$$

This function promises to rank models higher that represent a good consensus of all models.
Again, this function was evaluated with the help of the actual case IDs of the underlying process. We generated a set of model instances with different probability matrices and sorted them in a list with the fitness function. The result was a sorted list. For each list index we calculated the matrix' fitness according to the G-Score. This was done 1000??? times. In figure (insert figure) the G-Scores of matrices at each list index can be seen. This result shows that the sorting with the summed trace probabilities ranks models with a high fitness to the process higher than unfit models. This shows that this fitness function is a good prospect for a fitness approximation function, as it can approximately sort models according to their fitness.

Average G-Score Another approach that is quite similar to the summed trace probabilities is the use of the metric G-Score itself. Until now, the G-Score was always used to evaluate models against the ground truth; that is the actual process. Now one model instance is evaluated against another model instance, of which both have no claim to have a high fitness

$$f(m, M) = \frac{1}{|M|} \cdot \sum_{m_i \in M} G(m, m_i) \tag{5}$$

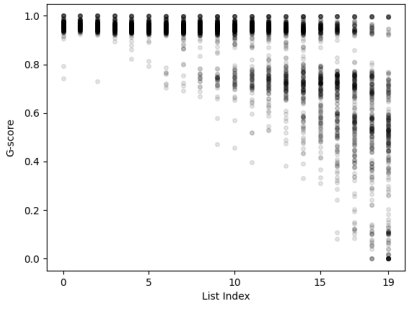

Fig. 4. G-Score - Trace Probability Comparison on Models

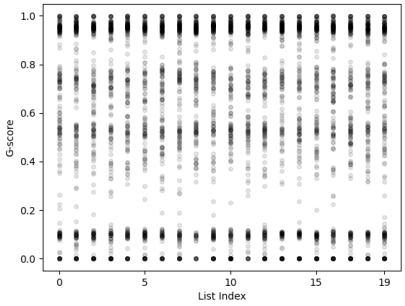

Fig. 5. Random weights

6 Evaluation

6.1 Experimental Setup

6.2 Results

- base vs genetic on the data set (one event log)
- base vs genetic on the data set (multiple event logs)
- base vs genetic on the data set with random init instead M+
- base vs genetic with loops
- base vs genetic on larger event log (3000 sources instead of 300)

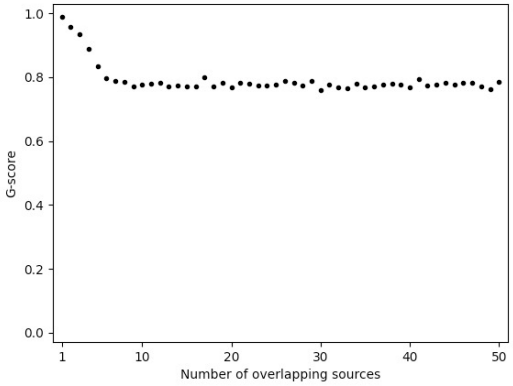

Fig. 6. Average G-score from the genetic extension.

7 Discussion

8 Conclusion

References

1. Bayomie, D., Helal, I.M., Awad, A., Ezat, E., ElBastawissi, A.: Deducing case ids for unlabeled event logs. In: International Conference on Business Process Management. pp. 242–254. Springer (2016)
2. Ferreira, D.R., Gillblad, D.: Source code to accompany the paper: Discovering process models from unlabelled event logs [3]. `http://web.ist.utl.pt/diogo.ferreira/mimcode/`
3. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs. In: International Conference on Business Process Management. pp. 143–158. Springer (2009)

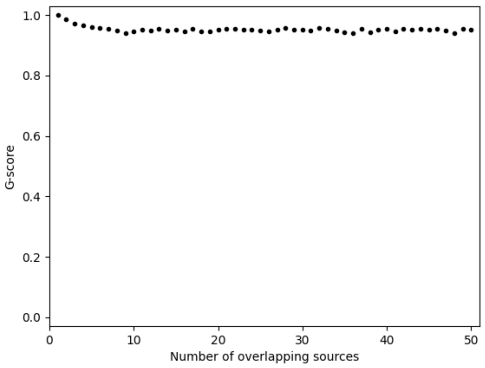

Fig. 7. Average G-score from the genetic extension.

4. Helal, I.M., Awad, A., El Bastawissi, A.: Runtime deduction of case id for unlabeled business process execution events. In: 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA). pp. 1–8. IEEE (2015)
5. Pourmirza, S., Dijkman, R., Grefen, P.: Correlation mining: mining process orchestrations without case identifiers. In: International Conference on Service-Oriented Computing. pp. 237–252. Springer (2015)
6. Vanneschi, L., Poli, R.: 24 genetic programming–introduction, applications, theory and open issues (2012)