Genetic Correlation Discovery for Unlabeled Event Logs

Pascal Schulze and Anjo Seidel

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

Abstract. In the context of creating event logs for process mining, the correlation of events by case identifiers to traces is essential. Unlabeled event logs do not provide any case notion. Therefore, more work is required to make a case notion available. We propose a working genetic extension to an already laid out approach building on an iterative expectation-maximization algorithm with greedy components and strong assumptions about the process. The extension is based on the techniques selection, reproduction, and mutation of multiple simultaneously trained solution candidates to come closer to the global optimum with each generation. The results show that the extension has improvement potential in terms of accuracy and of disposing some of the made assumptions. By reusing a defined model evaluation function, we address a main issue in this paper that is the necessity of a fitness function in order to rank solution candidates during the genetic procedure. The presented approach leaves room for improvements and provides the basis for a broad field of further research.

Keywords: process mining · process discovery · correlation discovery · genetic mining · unlabeled event logs

1 Introduction

Process Mining is a powerful technique to extract process models from process unaware systems with the help of event logs [1,11]. Event logs can be generated by first extracting data from the system and then correlating them to traces, which are a case representation of a process. In the end events are abstracted in order to depict only valuable information [8]. The correlation of events is therefore indispensable for process mining.

In many approaches, events are correlated by attributes that provide a case notion. Discovering the correlation of events is especially hard if the only available information is an unlabeled event log. Unlabeled event logs only provide the chronological order of events, from which only the event type is known. Current approaches try to solve this problem with probabilistic approaches. Many are based on the work of Ferreira et al. [6], which provides an iterative expectation-maximization procedure to estimate case IDs. Although this approach provides usable results and is the base of a lot of further research, it is bound to many

assumptions towards the underlying process. Furthermore, the greedy component of this approach promises to be open to improvements.

The genetic programming paradigm promises to solve problems related to pattern recognition [9]. Cases within a process can be seen as a pattern of events. A genetic approach can often be used to find the global optimum instead of only finding the local optimum, like greedy algorithms do in some cases. This can be achieved by applying principles from nature and Darwinism. This includes the selection, reproduction, and mutation of solution candidates [12].

This paper introduces an extension to the work of Ferreira et al. by introducing a genetic programming paradigm to solve the correlation problem. It aims to improve the present algorithms' results and to overcome some of the made assumptions. A big focus lies on the finding of an appropriate fitness function, which is needed for ranking and selecting solution candidates during the genetic procedure.

This results in the following papers structure. Section 2 presents work related to this paper while Section 3 describes the background of our research. In Section 4 the genetic extension will be explained while Section 5 provides the indispensable fitness function. The applicability of our findings will be evaluated and discussed in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

Several techniques to facilitate event correlation on unlabeled event logs have been developed already. These thought out approaches deduce the case identifiers for events by using different concepts and often rely on the existence of domain knowledge.

In 2009 Ferreira et al. [6] already came up with a zero-knowledge approach that 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. Further work was done in 2015 when Pourmirza et al. [10] published their idea of a proverbial correlation miner. The concept behind that miner is a merging, clustering, and aggregation process on the different cases in the event log to eventually generate a suitable orchestration model based on assigned identifiers. However, their miner requires a timestamp for every event to be applicable. As well in 2015, Helal et al. [7] and in 2016 Bayomie et al [2] proposed techniques that 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 for the unlabeled characteristic with further knowledge to finally come up with a derived labeled event log.

To the best of our knowledge, we cannot provide any related work to the topic of a genetic approach to solve the correlation problem on unlabeled event logs, neither with nor completely without any further background knowledge used as input. Thus, the genetic extension presented in this paper is part of a research gap in this field.

3 Background

In this Section, the general concept of genetic programming is introduced. Afterwards, we will give a brief insight into the approach presented by Ferreira's et al. [6][1], which is the core of our genetic extension. Furthermore, the test environment as well as the accuracy which is going to be used for algorithm accuracy comparisons will be defined.

3.1 Genetic Programming Paradigm

The concept of genetic programming comes from nature itself. It is an evolutionary approach where a computational program will explore the possible solution space with the help of a defined quality or fitness function. It evolves a set of solution candidates, instead of only training one, by mimicking the basic principles of Darwinian evolution [12]. After training a set of solution candidates for one epoch, they are evaluated and ranked regarding their individual fitness. Only the best-ranked candidates will be considered in the further process. This is called *selection*. The selected candidates will be chosen to *reproduce* the next generation of offsprings. The next epoch will be processed with the highest-ranked candidates and their offsprings. Thereby, the characteristics and features of the best individuals will be transferred to the next generation while weak properties will be neglected. As in nature, *mutations* can slightly change the random properties of individuals in the next generation in order to support more variance.

3.2 Case ID Estimation

The problem of correlating events, of which only the chronological order of occurrences is known, to event logs was tackled by Ferreira's et al. [6]. Here, a chronological sequence of event labels is taken as input – the input symbol sequence. These labels represent a specific event type and provide no further information. 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 procedure, a probabilistic method is presented and visualized in figure 1.

The correlation of events is derived from a probability matrix, which holds the probabilities for each sequence of two following events. As the starting point, the probability matrix is initialized by estimating it from the symbol sequence. The initial matrix M^+ computes the probability of a transition between two events by the number of direct successorships of those two events in relation to their overall occurrences. Thus, "consistent behavior will stand out with stronger transition probabilities than the spurious effects of random interleaving" [6]. With that probability matrix, traces from the input symbol sequence can now be estimated. Here, the algorithm acts greedy and therefore indicates its first weakness. It creates a new trace for each start event. All other events get assigned to the one existing trace, for which the last event has the highest probability of being

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

the predecessor of the current event. As a result, the estimated traces are subject to the initial probability matrix. In order to reduce the subjectivity and accordingly improve the accuracy of the initial matrix, the probability matrix can be recomputed from the event's successorships in the previously estimated traces. The new probability matrix now holds more precise probabilities than M^+. The iterative procedure of estimating traces based on a matrix and computing the matrix from the previously estimated traces is repeated until the predicted traces are not longer underlying changes. At that point, the last estimated traces are the output of this algorithm. Each of those estimated traces represents one case, for which a case ID can be assigned. The resulting event log can then be used further.

3.3 Base Accuracy

In order to evaluate the results of the base algorithm, Ferreira et al.[6] provided the G-score (equation 1) as an evaluation function. It compares two models m_1 and m_2, of which both have a set of traces and different probability distributions p_{m_1} and p_{m_2} for those traces. For every model m_1 the similarity to another model m_2 is evaluated by computing the geometric mean of both distributions. Thus, we get the true labeling accuracy of each predicted model.

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

For the evaluation, the input symbol sequences were generated randomly from a previously defined process, which provides a comparison with the G-Score. The accuracy of the provided python implementation [5] converges against an average G-score of around 0.82[2] "for input event logs with 300 sources and a varying number of overlapping sources [between 1 and 50 while] each point has been obtained by averaging over 1000 synthetically-generated logs" [6] visualized in figure 2.

4 The Genetic Approach

An assumingly big 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 that 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 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

[2] During our research we noted that the G-score has an asymptote of 0.82 instead of 0.6 as written in the paper by Ferreira et al. [6]. This found discrepancy was confirmed by the authors and can be traced back to a probably different used setup during their evaluation process. On the basis of this found discrepancy we will use our observed accuracy whenever comparing our extension with the base algorithm.

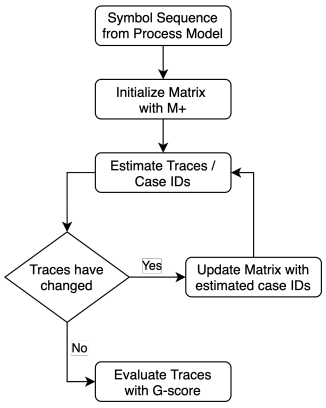

Fig. 1. Training's flow of the base algorithm [6]

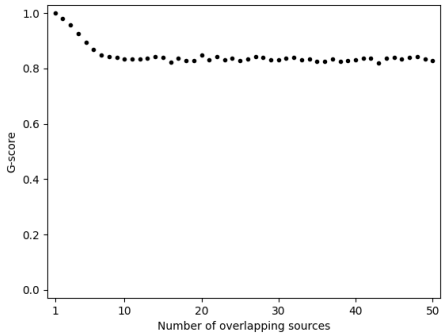

Fig. 2. Average G-score of the base algorithm [6]

algorithm with a genetic concept we can train multiple models simultaneously and connect them to slowly find a (local) optimum in a far wider range to overcome possible bad initialization by building on the fittest models in each epoch. According to the genetic programming paradigm described in section 3, we provided an extension for the base algorithm. This extension joins the concept of a genetic algorithm and the iterative expectation-maximization algorithm.

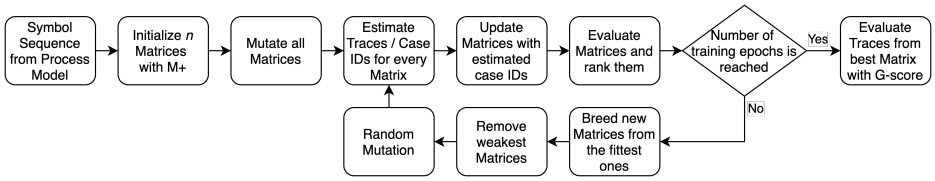

Fig. 3. Training's flow with the genetic extension.

As previously described, the genetic algorithm shown in figure 3 runs on multiple instances. Here, n probability matrices are estimated from an input symbol sequence x. Those probability matrices p 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 with a certain variety. Thus, n slightly different probability matrices are generated. Analog to the base algorithm, for each of the matrices p_{m_i} and the symbol sequence x, the traces y_{m_i} can be estimated. Each model instance m_i now consists of the input sequence x, the probability matrix p_{m_i} and estimated sequences y_{m_i}. Again, like in the base algorithm, we can then compute a new $\hat{p}_{m_i}$ from x and y_{m_i} 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 repro-

duction, two matrices can be merged by respectively building the average of all their values. That way, new probability matrices are generated which are superior in comparison with the weaker matrices. Complementary, the lowest-ranked and therefore weakest matrices can be excluded by replacing them with the newly generated ones. To provoke the possible creation of even fitter models, random mutations can be performed. For each of this improved set of matrices, a new generation of traces y_{m_i} can be estimated based on the new probability matrix $\hat{p}_{m_i}$ and the input sequence x.

The process of estimating traces – deriving new trace estimations and then new probability matrices that are ranked, selected, reproduced and mutated – is run repetitively. The algorithm's result in the end is highly influenced by the hyperparameters, which are the number of epochs, number of models, mutation probability and factor, and the reproduction factor. As soon as a given number of epochs is reached, the best ranked and therefore fittest matrix from the evaluation phase 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 simply for evaluation of the approach. The idea and implementation of the described algorithm are quite uncomplicated except for one crucial step – the fitness function. The challenge of finding an appropriate fitness function and the solution to this problem is described in section 5.

5 Fitness Functions

As already described in chapter 4 we need to rank our models by evaluating each interim solution to the problem. It has to be measured how "fit" a model is during the prediction process, where we assume we do not know the actual process model. As it is not straightforward to find an appropriate fitness function[3] for this use case, first the requirements and challenges are introduced. Afterward, different approaches are provided and the resulting candidate fitness functions are compared.

5.1 Requirements

A possible candidate for being a fitness or fitness approximation function has a structure as described in equation 2. Its result is a quantitative indicator of how fit a given input model m is. In our case a model m_i consists of a probability matrix p_{m_i} and the estimated traces y_{m_i} to a given symbol sequence x. The function needs to assign a model a higher value if the estimated traces and the probabilities of the matrix p_{m_i} represent the underlying process model better than other models. As we evaluate the overall algorithm with the G-Score, that is the G-Score of each model instance in comparison to the ground truth.

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

[3] It is also referred to as weight or evaluation function in our context.

The problem here is that we are working on an unlabeled event log – each model m_i is only one probabilistic prediction and thus does not contain any kind of quantifiable 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. In lack of additional information, X will be M, the set of all model instances m_i and their respective p_{m_i}, y_{m_i}, and l_{m_i}. The information held by M can be accessed without any restrictions during the genetic procedure.

It is also possible to pass other arbitrary information as X (e.g. additional symbol sequence(s) s_{m_i} if available), which would implicates new assumptions to the environment and algorithm. However, as the genetic approach is supposed to satisfy the same assumptions as presented by Ferreiera et al., we will only use M as seen in equation 3. For one fully genetic run of e epochs with n models the fitness function gets called $e \cdot n$ times. Therefore, a sufficiently fast function is necessary to minimize the possibility of being the computational bottleneck.

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

The comparison of those fitness functions requires an evaluation. As each function sorts a list of models, it needs to be determined, how well sorted this list is in the end. Therefore, for each candidate function, a list of more or less fit model instances of a fixed process model was taken as input. For the supposedly sorted models, the G-Score regarding the fixed process model could be computed. After sorting a thousand lists with the candidate function, the distribution of G-Scores, which resulted in different indices of the sorted list, gives insight into the function's ability to sort model instances according to the G-Score.

5.2 Candidate Functions

The limited information that is available as input is complicating the provision of a sufficient fitness function. As previously stated, only the information held in the considered model m_i and the other model instances in M can be used. The main idea, therefore, is to rank a model higher if it represents a good consensus of all models. It has to be noted that none of the model instances has any claim of correctness. Nevertheless, a model instance m_i can be evaluated against all the other model instances M. According to how fit this model instance is in comparison to every other model instance, the model can be ranked.

In the following, we will present different ideas and their performance in manners of sorting the models $m_i \in M$.

Alignment and Token Replay Alignment and Token Replay have been proven to be powerful techniques in the context of conformance checking [3,1]. As the fitness function, f aims to rank models $m \in M$ according to the conformance among each other, the use of those techniques seems to be promising.

Alignments of traces and models find the "closest path" of a trace through the model. The closer a path between trace and model, the better the fitness of the model in regards to the trace [3]. However, it has been shown that the alignment of

multiple traces at once is an NP-complete problem. Therefore, alignment seems to be unpromising for a fitness function in matters of run-time and complexity. Token Replay, on the other hand, can be managed for multiple traces. Originally, Token Replay calculates the fitness of a model in a Petri net or matrix representation by replaying a set of certain sequences of events with the Petri net. Missing and remaining tokens for a correct execution are set into relation with all consumed and produced tokens. This provides the model's fitness according to the set of traces [1].

Again, as the fitness function has to evaluate one model m_i against all other models $m_j \in M$ and the transitions of p_{m_i} 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. The results of the evaluation showed that sorting with these functions is equivalently bad as doing it randomly. This proved that these candidate fitness functions will not sort the models adequately for the genetic algorithm.

Summed 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, to compare the matrix of our model instance m_i against the estimated traces y_{m_j} of all other models $m_j \in M$.

The approach is based on some findings for our specific use case. First, the sum of the probabilities of all traces, which the matrix holds, is exactly one. The reason for that is that the matrix is computed based on these estimated traces and represents exactly their distribution of events. 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 our 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.

$$f(m_i, M) = \frac{1}{|M|} \cdot \sum_{m_j \in M} \left(\sum_{y \in y_{m_j}} p_{m_i}(y) \right) \tag{4}$$

This function promises to rank models higher that represent a good consensus of all models. In figure 4 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 higher than unfit models. Especially bad models are almost always ranked very low, while mainly good models are ranked high. This shows that this fitness function is a good candidate 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. The G-score can not only be

used for evaluating a model against the ground truth. But also to evaluate a model against any other model. That way it is possible to compare one model instance m_i against all other models $m_j \in M$.

In order to acquire a good consensus of all models, the model has to be chosen, which has the best average G-score in comparison to all other model instances. Therefore this candidate fitness function ranks all models according to the average of the G-scores for each other model instance. f takes a model instance m_i and all models $m_j \in M$ as input. It then calculates the G-score for each pair of traces y_{m_i} and y_{m_j}, which was already defined in 1. Then the average of all computed G-scores is computed.

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

This alternate use of the G-score as a fitness function turned out to provide great results.The results of the evaluation can be seen in figure 5. It shows an even better sorting of models than the *summed trace probabilities*. Especially in the front of the sorted list it is very likely to have only models with a great fitness, while the less fit models are all ranked lower.

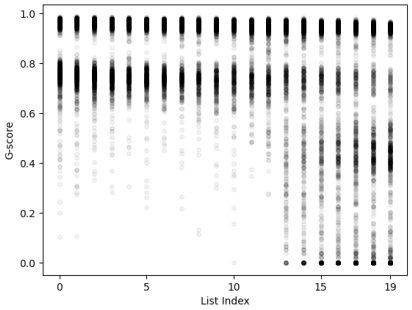

Fig. 4. Overall Trace Probabilities*

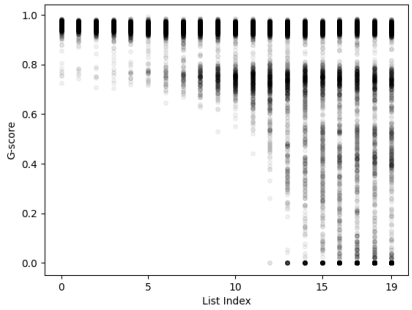

Fig. 5. Average G-score*

*Distribution of the true G-score of 1000 sorted sets with 20 independent models each generated from synthetically created event logs.

5.3 Comparison

Of the three implemented candidate fitness functions only two could provide a usable approximated sort of the model list according to the actual G-score of each model while the token replay implementations provided only a fitness assignment per model equivalent to random. The *summed trace probabilities* function was able to rank unfit models lower than mediocre and good ones. But, it still lacked in distinguishing between very fit models and slightly inferior ones which is not necessarily a criteria for exclusion, since the overall approximation is definitely usable. The *average G-score* function is able to rank models with a low fitness very low and to sort mainly the very fit models at the front of the list even better. This

leads to the conclusion that the average G-score function provides an adequate fitness approximation function. As a result, we were able to implement the genetic extension with a reuse of the average G-score as a fitness approximation function.

6 Evaluation and Discussion

In this section, we will provide an evaluation of the genetic extension and a comparison with the base algorithm. The results will be interpreted and discussed. Additionally, possible improvements to the genetic algorithm will be introduced. Those improvements and further uses of our approach constitute the future work that the genetic extension implies.

6.1 Experimental Comparison

With the indispensable fitness approximation function that could be found in section 5 it was possible to implement the genetic algorithm described in section 4. This algorithm can estimate case IDs by correlating traces of unlabelled event logs. It claims to solve the same problem as the base algorithm presented by Feirrera et al. and it gives reason to believe to have superior results. The in and output of the implementation are the same as provided by Feirrera et al – no further information was used during the estimating or evaluation process. The proposed genetic extension is parameterizable and thus multiple various hyperparameter configuration sets were tested in order to find a suiting one that provides the best results in regards to accuracy. The experimental setup that showed the best results will be explained and evaluated.

The result of the described experiment is a G-score asymptote of around 0.79 as shown in figure 6. It is apparent that the results of the genetic algorithm are similar to the base algorithm of 0.82, but yet slightly inferior. These results of our genetic approach can be interpreted in two ways. *Firstly*, it can be argued that the base algorithm, as provided by Feirrera et al., is already the best approximation to the best possible solution for the described problem that exists. This would imply that there cannot be found any better algorithm or extension around it. Yet, this would still need to be proven. Or *secondly*, we have not found a configuration of hyperparameters, which results in higher accuracy and thus provides a superior genetic algorithm. If it cannot be proven that the base algorithm is the best computation of traces, then such a configuration should exist. Yet, such a configuration still needs to be found.

The problem here is that only a large number of executed experiments lead to significant results due to a not negligible variance in the accuracy of multiple experiments. As the genetic approach runs many model instances simultaneously the computation time of the algorithm is significantly higher than the base algorithm's. A larger number of model instances in the algorithm will lead to probably better models, which will improve the results. As every epoch improves the set of models by selection and reproduction, a higher number of epochs will improve the outcome as well. Both come at the expense of a yet polynomial but still significant rise of computation time. Other hyperparameters that were previously

mentioned and can be tuned are the mutation rate and mutation factor, as well as the reproduction rate. Which configuration of those hyper-parameters leads to better results than the base algorithm, still needs to be evaluated. Finding a suiting hyperparameter configuration is therefore associated with the need for more time expensive experiments and their evaluations. Further and time-consuming work will be needed to solve this issue and answer the open questions above.

6.2 Outlook

Not only the hyperparameters but also the genetic approach itself can be extended and improved. The genetic programming paradigm provides many different strategies in the selection and reproduction step. It needs to be seen, whether different strategies like tournament selection [4], initialization points, and more can increase the genetic results as well.

One possible way to extend the genetic algorithm and the underlying concept from a non-algorithmic perspective is the usage of multiple symbol sequences as input instead of a single one. With multiple input event logs, it is possible to depict more behavior and characteristics. This is especially crucial if the process has more than just one start or end event. Here, the base algorithm has one implicit but important assumption: The initial Matrix M^+ estimates the probability of the start event by the occurrences, where an event is the first event of the symbol sequence. The first event is therefore always the only start event and has always a 1.0 probability. Analog, exactly one end event is predicted, which is always the last event of the symbol sequence. Thus, the start and end events of the symbol sequence are always the estimated start and end events for all traces and therefore of the predicted process model. Multiple input symbol sequences could therefore be used to depict such a behavior of a process. As the genetic algorithm initializes multiple matrices, those symbol sequences can easily be used for creating different model instances. Those matrices are then selected and reproduced. The reproduction can then lead to matrices that fit a process with more than one start and end event well.

This assumption of the base algorithm can now be overcome, but it still has to be evaluated in the future. For the same process, however, it shows great results, as shown in figure 7. Here, 10 different symbol sequences were used with 300 traces each as input for each of the 1000 separate experiments to average the G-score. It has to be noted, though, that only an overall increased number of traces results in a comparable state to the base algorithm. But additionally, it can be said that the genetic algorithm is now able to predict multiple start and end events. As well, it is giving more overall possibilities to soften or remove assumptions by improving the accuracy with the intelligent decision making of removing bad models and only continuing with the fittest ones. Thus, the approach itself can be improved and extended but already promises to yield superior results for different hyperparameter configurations.

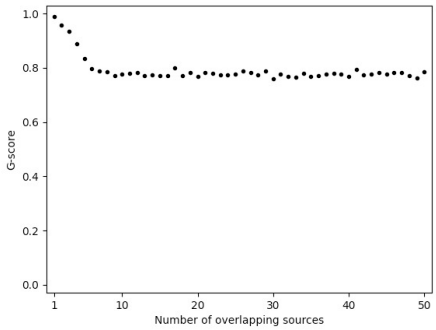

Fig. 6. Input: one symbol sequence*

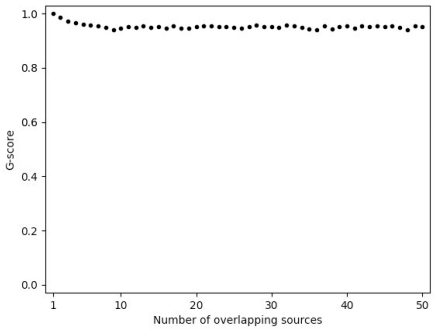

Fig. 7. Input: 10 symbol sequences*

*Average G-score from the genetic extension on different numbers of symbol sequences as input. Each point was obtained from 1000 symbol sequence with 300 sources each and the relating number of overlapping.

7 Conclusion

In the context of event log generation for process mining, the task of correlating events to traces is crucial. This is necessary to provide case IDs that are needed for every event log. Solving this correlation problem is especially hard if there is no further information than the order of occurrences of events – so-called unlabeled event logs. Iterative approaches with greedy components have tackled this problem but still leave room for improvement.

We have shown that a genetic approach for this use case works. This is not trivial, as the fitness function in the genetic approach can not rely on any ground truth during the estimation process. The genetic algorithm acts iteratively. In contrast to the base algorithm, it runs each epoch on many model instances instead of only one. Additionally, it selects only the fittest model instances and reproduces them. For the selection, we were able to find a fitness approximation function that can evaluate and rank model instances, which is necessary for a genetic algorithm.

It could be shown that such a fitness function, which evaluates the consensus of all model instances without any further information, can rank models according to the G-Score by assuming a model is better if it covers more of the other model's estimation for the same process model. This is crucial if no ground truth is available. Although we could prove that the current algorithm leads to approximately equal results as the present approach, these results are still open for improvements. Still, with this new genetic approach, a whole new range of possible extensions to the current state of research emerges. Now, it is possible to overcome old assumptions and extend the application of genetic process discovery to various business processes.

References

1. Wil van der Aalst, M.: Process mining. discovery, conformance and enhancement of business processes (2011)
2. 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)
3. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking. Springer (2018)
4. Fang, Y., Li, J.: A review of tournament selection in genetic programming. In: International Symposium on Intelligence Computation and Applications. pp. 181–192. Springer (2010)
5. Ferreira, D.R., Gillblad, D.: Source code to accompany the paper: Discovering process models from unlabelled event logs [6]. `http://web.ist.utl.pt/diogo.ferreira/mimcode/`
6. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs. In: International Conference on Business Process Management. pp. 143–158. Springer (2009)
7. 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)
8. Kiarash Diba, Kimon Batoulis, M.W.M.W.: Extraction, correlation, and abstraction of event data for process mining (2019)
9. Koza, J.R.: Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems, vol. 34. Stanford University, Department of Computer Science Stanford, CA (1990)
10. 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)
11. Van Der Aalst, W.: Process mining: Overview and opportunities. ACM Transactions on Management Information Systems (TMIS) **3**(2), 1–17 (2012)
12. Vanneschi, L., Poli, R.: 24 genetic programming–introduction, applications, theory and open issues (2012)