

Inductive Miner with Clustering of Fall-Through Sublogs

Wallesson Cavalcante da Silva

Programa de Pós-graduação em Computação, UFC – Campus Quixadá
Av. José de Freitas Queiroz, 5003 – Cedro – Quixadá – Ceará
wallessondm@gmail.com

Críston Pereira de Souza

Programa de Pós-graduação em Computação, UFC – Campus Quixadá
Av. José de Freitas Queiroz, 5003 – Cedro – Quixadá – Ceará
criston@ufc.br

ABSTRACT

Extracting interpretable business process models from event logs of poorly structured (spaghetti) processes is challenging. The Inductive Miner algorithm is a popular process mining algorithm. However, it tends to produce fall-through nodes when applied to spaghetti processes, resulting in a model with many stacked activities, which makes it harder to interpret the underlying process. In this work, we propose an adaptation to the Inductive Miner algorithm, called IM_Cluster, that clusters the traces of the event log associated with fall-through nodes, aiming to produce fewer occurrences of stacking of activities. We also devised a new simplicity metric that can capture the occurrence of stacking of activities, and the IM_Cluster relies on this metric. An empirical evaluation is conducted to compare the proposed algorithm with the original Inductive Miner and the ActiTraC algorithm. The results show that the proposed algorithm produces fewer occurrences of stacking of activities than the compared algorithms.

KEYWORDS. Process mining. Clustering. Spaghetti processes.

Paper topics (indicate in order of PRIORITY the paper topic(s)):

IC – Computational Intelligence

AD&GP – OR in Administration and Production Management

POI – OR in Industry

OA – Other applications in OR

BDA – Big Data e Analytics

1. Introduction

Understanding how a business process works is important since it allows, for example, identifying inefficiencies and wrong behaviors. A large amount of data is usually collected from information systems and stored in event logs. Extracting a business process model from an event log is known as *process mining* [Van Der Aalst, 2016]. The devising of process mining algorithms is an active area of research, and several algorithms have been proposed in the literature. The Inductive Miner [Leemans et al., 2013] is one of the most popular process mining algorithms, and it has provided good solutions in comparative studies [Augusto et al., 2018].

Processes with self-directed behavior of customers and professionals are typically poorly structured [Van Der Aalst, 2016]. These processes are known as spaghetti processes and are challenging to process mining algorithms. The Inductive Miner, for example, tends to produce many fall-through nodes in the resulting process tree when applied to spaghetti processes. A process tree is a hierarchical way to represent business process models where the internal nodes are operators (ex., exclusive choice, parallel composition, and sequential composition) and the leaves are activities. A fall-through occurs when the Inductive Miner cannot choose an appropriate operator for some node in the tree, and some special treatment is needed to handle this situation. For example, it may decide to put all activities under a single exclusive choice operator, which means that the business process may follow to any of these activities. When visualizing business process models with fall-through nodes, it is common to see many activities stacked in a single region, as illustrated in Figure 4, which makes the process harder to understand.

In this work, we propose an adaptation to the Inductive Miner algorithm that clusters the traces of the event log associated with fall-through nodes. In this way, the resulting models tend to present fewer occurrences of stacking of activities. We also devise a new simplicity metric that can capture the occurrence of stacking of activities, and the IM_Cluster relies on this metric to decide where in the process tree the change is worth. Besides, an empirical evaluation is conducted to compare the proposed algorithm with the original Inductive Miner and with the Active Trace Cluster (ActiTraC) algorithm [De Weerd et al., 2013].

The remainder of this paper is organized as follows. Section 2 presents some related works for the problem of simplifying business processes. Section 3 details the proposed algorithm. Section 4 presents the empirical evaluation, including the proposed simplicity metric, a description of the experimental datasets, and a discussion of the results. Finally, Section 5 concludes the paper and presents directions for future works.

2. Related Works

Several studies have addressed the problem of simplifying and grouping process models to improve the understanding and management of business processes. The Fuzzy Miner [Van Der Aalst and Gunther, 2007] simplify complex models using cartography concepts such as aggregation, abstraction, and emphasis. The approach of [Song et al., 2008] introduces a methodology for grouping similar traces using trace profiles to characterize and compare traces, highlighting different types of profiles, such as transition, case and event attributes, and performance. They use four clustering algorithms to group traces and improve process mining models, including K-Means, Quality Threshold, Agglomerative Hierarchical Clustering, and Self-Organizing Maps. The Active trace clustering (ActiTraC) approach [De Weerd et al., 2013] proposes a trace clustering technique based on active learning. They seek to group traces not only by similar behavior but also by their suitability to a specific process model. In [Wang et al., 2018], a clustering technique based on the tight alignment of traces is proposed, adapting two clustering strategies: Agglomerative Hierarchical Clustering and Spectral Clustering. This work aims to ensure the proper representation of essential activities in traces, using hierarchical flexibility and adaptive capability to provide a

comprehensive and accurate view of the process. The Skip Miner algorithm [Batista and Solanas, 2019] introduces a technique for simplifying process models that use an integrated simplification strategy, ignoring events based on a probability, balancing simplification and the quality of the models discovered. The Split Miner method [Augusto et al., 2019] is a technique for producing simple process models with low branching complexity, standing out for its high and balanced fitness and precision. The Common Event Identifier (IEC) framework [Tariq et al., 2021] aims to identify groups of event levels in business process logs. It is divided into four steps: log preparation, sequential rule extraction, residual correlation evaluation, and common event identification. The Clustering-Based Pattern Mining (CBPM) framework [Djenouri et al., 2021] uses clustering techniques to simplify pattern discovery in large datasets. The Approximate Inductive Miner framework (AIM) [Van Detten et al., 2023] automates the discovery of models with polynomial time complexity without the need for preprocessing or parameter input. The Interactive Trace Clustering [Neubauer et al., 2022] combines human knowledge with automated clustering algorithms to improve business process analysis.

3. Solution approach

Applying the Inductive Miner algorithm to spaghetti processes typically produces fall-through nodes, such as the one highlighted in Figure 1. In order to simplify the model and capture more characteristics of the underlying process, the algorithm *IM_Cluster* is proposed in this work and is detailed in Algorithm 1. The input is an event log L , which in this work is represented by a set of traces, and the output is a process tree. Initially, the *IM_Cluster* applies the Inductive Miner to construct a process tree T for L . Then, it identifies each fall-through node ft in T that does not satisfy the stop condition (function *FindAllFTs*). For each ft , the algorithm extracts two clusters L_1 and L_2 from the sublog¹ $ft.log$ of ft (function *LogClusters*), and applies the *IM_Cluster* recursively to each cluster, resulting in subtrees T_1 and T_2 . Then, the algorithm creates a new node v with children T_1 and T_2 connected by an XOR operator. The XOR operator means that the process instance must follow through exactly one of the children subtrees T_1 or T_2 . If the quality of v is better than that of ft , the algorithm replaces ft by v in T . The quality metric is defined in Section 4.1.2. Finally, the algorithm returns the updated process tree T .

Algorithm 1: *IM_Cluster*

Input: L : event log (set of traces).
Output: T : process tree.

```

1  $T \leftarrow \text{InductiveMiner}(L)$ 
2 for  $ft \in \text{FindAllFTs}(T)$  do
3    $L_1, L_2 \leftarrow \text{LogClusters}(ft.log)$ 
4    $T_1 \leftarrow \text{IM\_Cluster}(L_1)$ 
5    $T_2 \leftarrow \text{IM\_Cluster}(L_2)$ 
6    $v \leftarrow$  new XOR node with children  $T_1$  and  $T_2$ 
7   if  $\text{Quality}(ft) < \text{Quality}(v)$  then replace  $ft$  by  $v$  in  $T$ 
8 return  $T$ 
  
```

The *FindAllFTs* function is detailed in Algorithm 2. The input is the root node r of a process tree T , and the output is a set of fall-through nodes of T . If r is null or satisfies a stop condition, the function returns an empty list. If r is a fall-through node, the function returns a list

¹The Inductive Miner variant considered here applies some transformation to the event log after applying each operator. We call the *sublog* of a node u the transformed event log at node u .

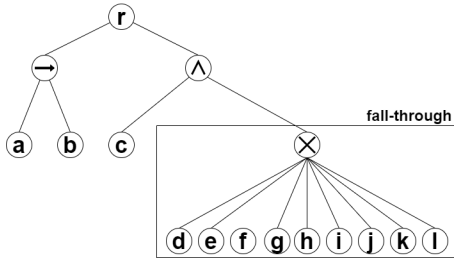


Figure 1: Process tree with *fall-through* node.

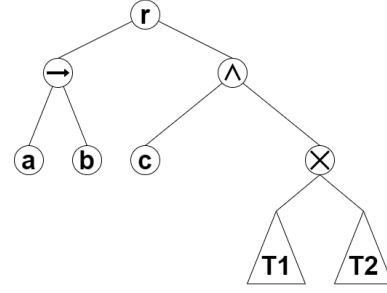


Figure 2: Replacement of the *fall-through* node.

containing only r . Otherwise, the function returns the union of the results of applying *FindAllFTs* to each child node v of r . Note that the returned fall-through nodes are roots of disjoint subtrees of T , since the recursion is not applied to descendants of these nodes. The stop condition occurs when the number of traces in the sublog of r is less than a predefined threshold m . This condition avoids the generation of small clusters that do not contribute to the simplification of the process model.

Algorithm 2: *FindAllFTs*

Input: r : root of process tree T .

Output: set of selected *fall-throughs* nodes of T

- 1 **if** r is null or $StopCondition(r)$ **then return** $[]$
 - 2 **if** r is a *fall-through* node **then return** $[r]$
 - 3 **return** $\bigcup_{v \in r.filho} FindAllFTs(v)$
-

Algorithm 3 describes the steps of the *LogClusters* function. The input consists of an event log L , and the output is the sublogs L_1 and L_2 resulting from the clustering of the traces of L . Initially, the traces of L are converted into strings and stored in S . Then, the Levenshtein distance matrix M [Levenshtein et al., 1966] is built for the strings of S . With the distance matrix M , the strings in S are clustered through the k -medoids algorithm [Kaufman and Rousseeuw, 1990]. Finally, the groups are converted back into traces to be returned as the sublogs L_1 and L_2 .

Algorithm 3: *LogClusters*

Input: L : event log (set of traces).

Output: L_1 and L_2 : partition of L (two clusters).

- 1 $S \leftarrow$ convert each trace in L to string
 - 2 $M \leftarrow$ compute the Levenshtein distance matrix of S
 - 3 $S_1, S_2 \leftarrow k\text{-medoids}(S, M)$
 - 4 $L_1 \leftarrow$ convert each string in S_1 back to the corresponding trace
 - 5 $L_2 \leftarrow$ convert each string in S_2 back to the corresponding trace
 - 6 **return** L_1, L_2
-

4. Empirical evaluation

This section presents the empirical evaluation of the IM_Cluster algorithm and compares it with the ActiTraC [De Weerd et al., 2013] and the Inductive Miner [Van Der Aalst, 2016]

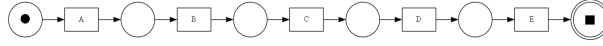


Figure 3: Example of Petri net with a sequence of places and transitions.

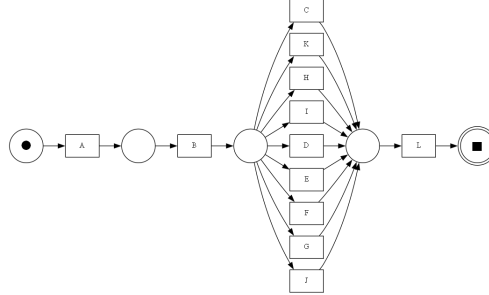


Figure 4: Example of Petri net with stacking transitions (activities).

algorithms. The comparison employs the ActiTraC implementation available in the ProM plugin [Van Dongen et al., 2005] and the Inductive Miner implementation available in the PM4Py library [Berti et al., 2023], both with default parameter values. The ActiTraC returns four process trees, which are joined in a single tree via the exclusive choice operator. The IM_Cluster algorithm is implemented in Python 3.8. The experiments were conducted on a virtual machine with an Intel Core i7-10700 (2.9GHz) processor and 64GB of RAM.

The employed experimental event logs are the BPI Challenges 2012 and 2020 [Van Dongen, 2012; González et al., 2020] since they are public and present poorly structured processes. The BPI Challenge 2012 event log contains 13,087 traces, 36 activities, and 262,200 events collected from a Dutch financial institute between October 2011 and March 2012, related to loan/overdraft requests. The BPI Challenge 2020 event log from the Eindhoven University of Technology refers to the reimbursement of international travel expenses, containing 6,449 traces, 34 activities, and 72,151 events collected between 2017 and 2018.

In the remaining of this section, Section 4.1 describes the metrics used to evaluate the algorithms and Section 4.2 presents and discusses the experimental results.

4.1. Evaluation metrics

Now, we describe the metrics used to evaluate the algorithms. The proposed simplicity metric tries to capture the decrease in model complexity by removing fall-through nodes. The overall model quality balances the precision [Blum, 2015] and the proposed simplicity metric.

4.1.1. Proposed simplicity metric

A common way to measure the model's simplicity is by considering the average degree of places of the generated Petri net model [Blum, 2015]. In order to penalize the stacking of activities produced by the occurrence of fall-through nodes, as illustrated in Figure 4, the proposed metric GP in Equation (1) squares of the input and output degrees of the places in a Petri net N . In the example of Figure 3, the places and transitions form a simple sequence, and the GP is equal to the average degree. In the example of Figure 4, on the other hand, we have a stacking of nine activities, and the GP is 33.6, much greater than the average degree (which has value 7.3).

$$GP(N) = \frac{\sum_{\text{place } i \text{ in } N} (\text{in-degree}_i^2 + \text{out-degree}_i^2)}{\text{number of places in } N}. \quad (1)$$

The Simplicity_GP metric, defined in Equation (2), is similar to the simplicity metric of Blum [2015], but it uses the GP value instead of the average degree of the places.

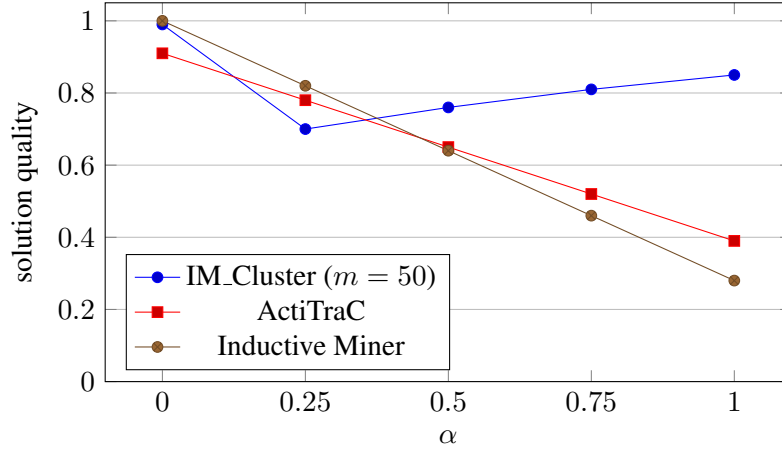


Figure 5: Solution quality varying α for the BPI Challenge 2012.

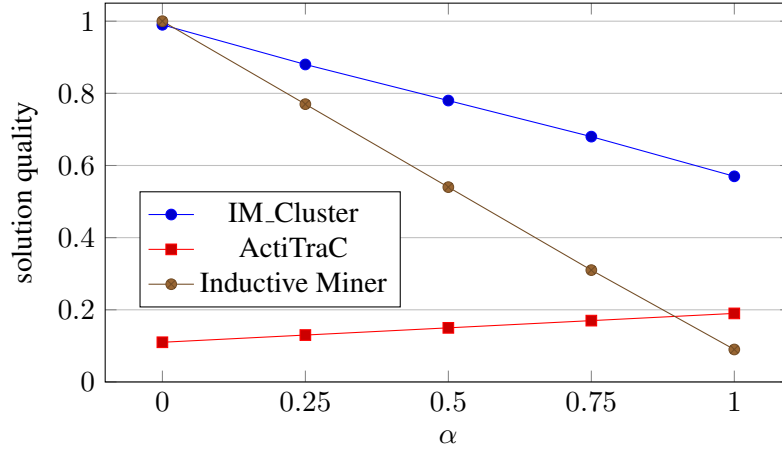


Figure 6: Solution quality varying α for the BPI Challenge 2020.

$$\text{Simplicity_GP}(N) = \frac{1}{1 + \max(\sqrt{\text{GP}(N)} - 2, 0)}. \quad (2)$$

4.1.2. Solution quality metric

The employed solution quality metric of a model is extracted from its corresponding Petri net N , and combines the Precision metric [Blum, 2015] and the Simplicity_GP (Section 4.1.1). Equation (3) describes the solution quality of a Petri net N . The parameter $0 \leq \alpha \leq 1$ controls the relative importance of the Simplicity_GP. The precision is computed for each trace t in the event log L , so the Quality metric considers the average precision of the traces in L . As both the Precision and the Simplicity_GP are values between zero and one, the Quality of a model is also in this range.

$$\text{Quality}(N, L, \alpha) = \alpha \cdot \text{Simplicity_GP}(N) + (1 - \alpha) \cdot \frac{\sum_{t \in L} \text{Precision}(N, t)}{|L|}. \quad (3)$$

4.2. Experimental results

The IM_Cluster is a recursive algorithm with two base cases: the considered node is either the root of a tree without fall-through nodes, or it is a fall-through node with a sublog having fewer

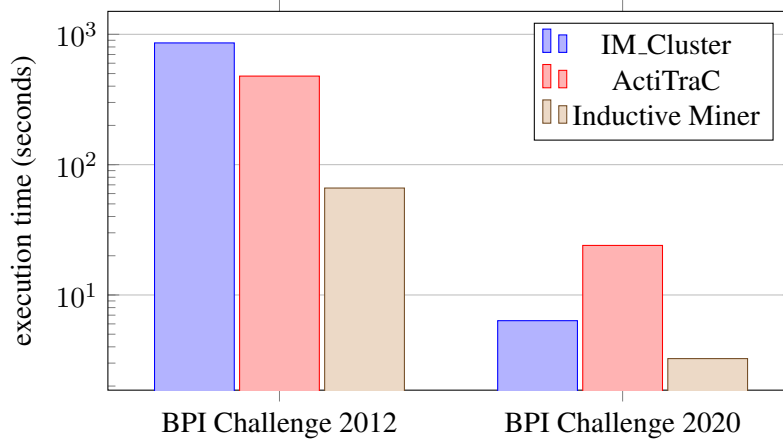


Figure 7: Comparison of execution times.

than m traces. The BPI Challenge 2020 event log has only one fall-through node in the process tree returned by the Inductive Miner, so there is no need to vary the parameter m . On the other hand, the BPI Challenge 2012 event log has several fall-through nodes, so we need to choose the value of the parameter m . By testing the values of m in the set $\{50, 100, 150, 200, 250, 300\}$, we found that the value $m = 50$ produces the best results for the Quality metric. The IM.Cluster algorithm is also stochastic, as it employs the k -medoids algorithm. Thus, in each experiment, we executed the IM.Cluster 10 times and collected the mean and standard deviation. However, we do not report the standard deviation in this section since the values were always negligible compared with the means. As the Quality metric depends on the parameter α , we considered each value for α in the set $\{0, 0.25, 0.5, 0.75, 1\}$ in all experiments.

Figures 5 and 6 present the results of the Quality metric varying the parameter α for the BPI Challenges 2012 and 2020, respectively. Note that the Quality metric is the average precision when $\alpha = 0$ and the Simplicity_GP when $\alpha = 1$. As the solutions of ActiTraC and Inductive Miner are not affected by the parameter α , the Quality metric is a linear function of the parameter α , interpolating between the average precision and the Simplicity_GP. On the other hand, the behavior of the IM.Cluster depends on the parameter α , since the algorithm decides to replace a fall-through node based on the Quality metric, so the Quality of the resulting model may not be a linear function of α .

For the BPI Challenge 2012 event log, the IM.Cluster has Quality similar to the ActiTraC and Inductive Miner for $\alpha < 0.5$. However, after this point, it has a much better Simplicity_GP, resulting in better Quality. The ActiTraC and Inductive Miner algorithms have similar results for the BPI Challenge 2012 event log, with the Inductive Miner having a slightly higher average precision and the ActiTraC having a slightly higher Simplicity_GP. For the BPI Challenge 2020 event log, the IM.Cluster has Quality better than the other algorithms for practically all values of α . The Inductive Miner has a low average precision and low Simplicity_GP for the BPI Challenge 2020 event log, and the ActiTraC has a low Simplicity_GP but a high average precision.

Table 1 presents the number of traces in the clusters produced by ActiTraC when applied to the event logs of the BPI Challenges 2012 and BPI Challenge 2020. The “Other” cluster contains the traces that were not well adjusted to the other clusters, being the largest cluster in the BPI Challenge 2012 and the second largest cluster in the BPI Challenge 2020, indicating that the algorithm failed to categorize a large part of the traces. In addition, half of the other clusters have less than 10% of the total traces. These results indicate that the ActiTraC algorithm, or the default parame-

Table 1: Number of traces in the clusters provided by the ActiTraC.

Cluster	BPI Challenge 2012	BPI Challenge 2020
Cluster 1	3.430 (26,2%)	3.431 (53,2%)
Cluster 2	1.872 (14,3%)	502 (7,78%)
Cluster 3	271 (2,07%)	320 (4,96%)
Other	7.514 (57,41%)	2.196 (34,05%)

ter in ProM's implementation, could not capture relevant subprocess structure for our experimental event logs.

Figure 7 presents a comparison of the execution times of the algorithms for the BPI Challenges 2012 and 2020 event logs. The BPI Challenge 2012 event log has about twice the number of traces the BPI Challenge 2020 event log, so the execution time is much higher for all algorithms. In both event logs, the Inductive Miner has the lowest execution time. Recall that the IM_Cluster executes the Inductive Miner for each recursive call, including a call with the complete event log as input. For the BPI Challenge 2012 event log, the IM_Cluster has the highest execution time (80% higher than the ActiTraC). On the other hand, the IM_Cluster is almost four times faster than the ActiTraC for the smaller BPI Challenge 2020 event log.

4.2.1. Inspection of resulting models

Figures 8, 9, and 10 present the Petri nets obtained by each algorithm for the BPI Challenge 2012 event log, and figures 11, 12, and 13 present the Petri nets obtained by each algorithm for the BPI Challenge 2020 event log. For the IM_Cluster, the model was obtained with $\alpha = 0.5$, which is the scenario with balanced average precision and Simplicity_GP. Note that the stacking of activities appears in the ActiTraC and Inductive Miner models but not in the IM_Cluster models. However, the IM_Cluster model for the BPI Challenge 2012 event log has a more complex structure, with many more transitions and places.

5. Conclusions and future works

Poorly structured (spaghetti) processes are common in real-world applications and are challenging for process mining algorithms. The Inductive Miner is a very popular algorithm for process mining, but it tends to stack activities in the resulting models when applied to spaghetti processes. The stacking of activities results from the fall-through nodes, which occurs when the Inductive Miner is unable to determine which operator to apply. In this work, we tackle the problem of stacking activities by clustering the traces in the sublog of fall-through nodes. We call this algorithm IM_Cluster. This work also proposes a simplicity metric called Simplicity_GP, which better captures the presence of fall-through nodes in process trees. This metric is then employed to guide the IM_Cluster algorithm.

An empirical evaluation was conducted using the BPI Challenge 2012 and 2020 event logs, comparing the IM_Cluster with the Inductive Miner and ActiTraC algorithms. The results show that the IM_Cluster was able to reduce the stackings in the built models, keeping an average precision that was not worse than that of the other algorithms.

We now discuss the limitations of this work and suggest future works. The IM_Cluster algorithm is slower for the largest event log, demanding about 80% more time than the ActiTraC. So it may be worth investigating the use of more efficient clustering algorithms and Inductive Miner variants, taking into account that only Inductive Miner variants that can handle sublogs can be used since the IM_Cluster clusters these sublogs. Another limitation of this work is that the IM_Cluster algorithm was optimized for the value of m , and the parameters of the alternative algorithms were not, making the comparison somehow unfair. Besides, the IM_Cluster is aware of the Quality metric



Figure 8: Petri net produced by the IM.Cluster for the *BPI Challenge* 2012.

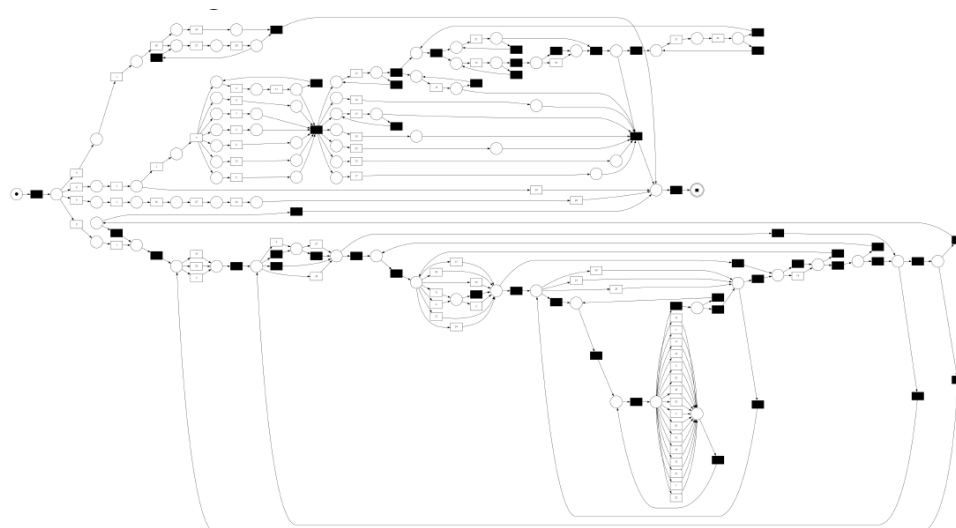


Figure 9: Petri net produced by the ActiTraC for the *BPI Challenge* 2012.

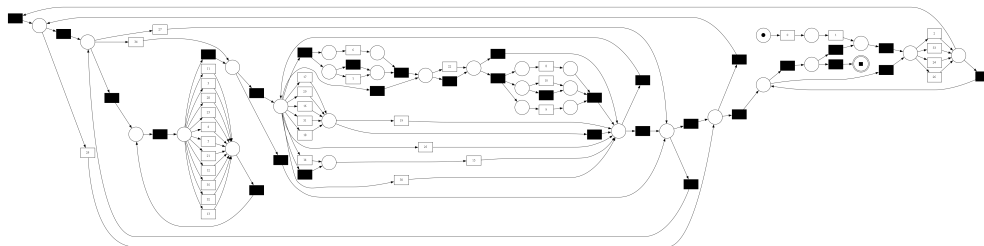


Figure 10: Petri net produced by the IM for the *BPI Challenge* 2012.

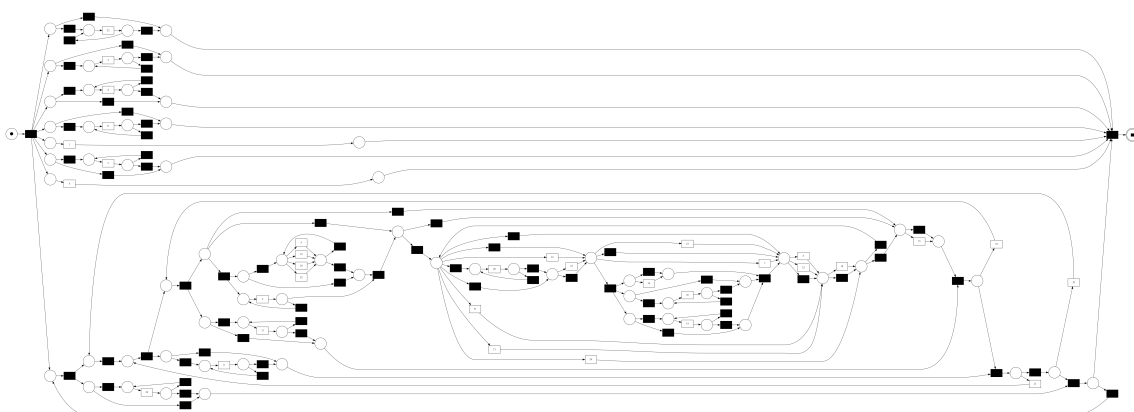


Figure 11: Petri net produced by the IM.Cluster for the *BPI Challenge* 2020.

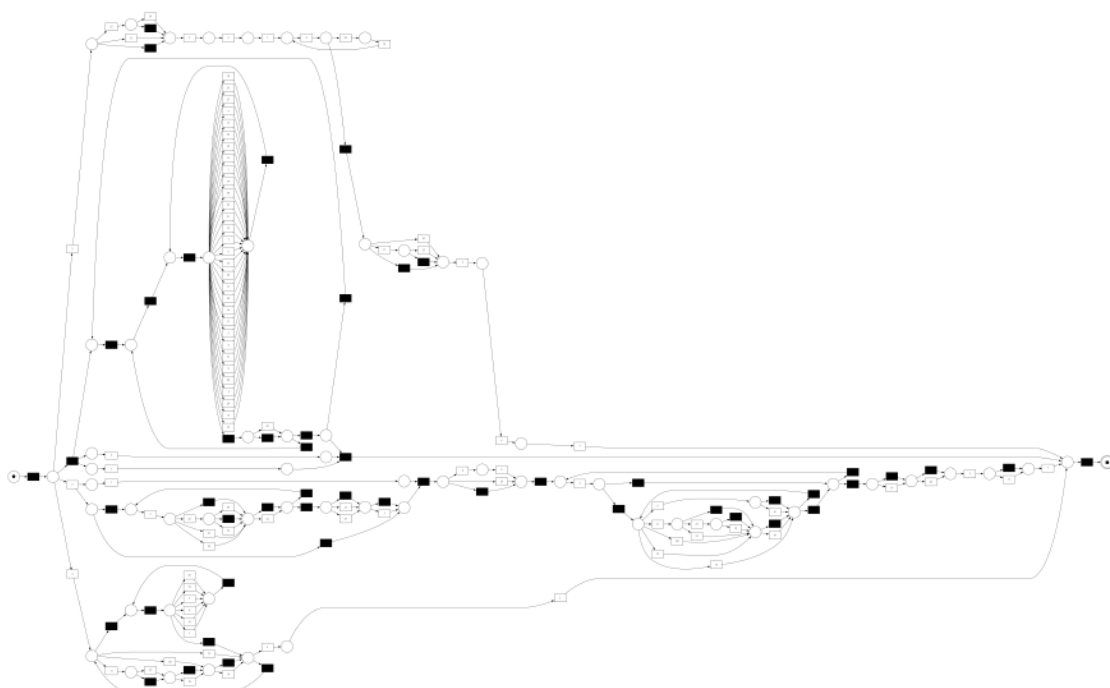


Figure 12: Petri net produced by the ActiTraC for the *BPI Challenge* 2020.

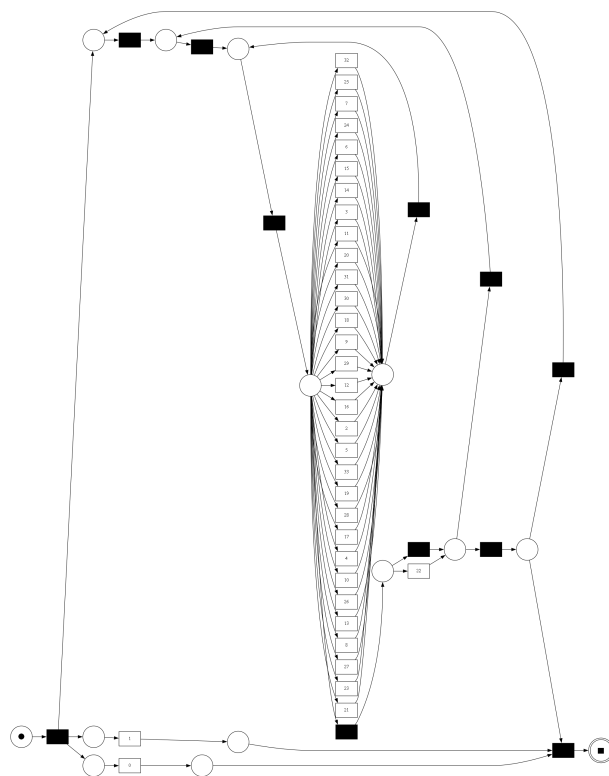


Figure 13: Petri net produced by the IM for the *BPI Challenge 2020*.

employed in the comparison, while the other algorithms are not. Finally, the IM.Cluster algorithm cannot control the resulting model's size, which may make the resulting models hard to interpret. So, it is worth investigating ways to control the size of the models provided by the IM.Cluster, such as employing alternative stop conditions.

References

- Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F. M., Marrella, A., Mecella, M., and Soo, A. (2018). Automated discovery of process models from event logs: Review and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 31(4):686–705.
- Augusto, A. et al. (2019). Split miner: Automated discovery of accurate and simple business process models from event logs. *Knowledge and Information Systems*, 59:251–284.
- Batista, E. and Solanas, A. (2019). Skip miner: Towards the simplification of spaghetti-like business process models. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, p. 1–6. IEEE.
- Berti, A., van Zelst, S., and Schuster, D. (2023). PM4Py: a process mining library for Python. *Software Impacts*, 17:100556.
- Blum, F. R. (2015). Metrics in process discovery, tech. rep., tech. rep. tr/dcc. Technical Report TR/DCC, Computer Science Dept., University of Chile.

- De Weerd, J., vander Broucke, S., Vanthienen, J., and Baesens, B. (2013). Active trace clustering for improved process discovery. *Transactions on Knowledge and Data Engineering*, 25(12): 2708–2720.
- Djenouri, Y. et al. (2021). Exploring decomposition for solving pattern mining problems. *ACM Transactions on Management Information Systems (TMIS)*, 12(2):1–36.
- González, L., Arenas, H., Lara, J., Rebolledo, F., and Ribbeck, K. (2020). BPI Challenge: Process Mining Analysis for Business Travel Reimbursement. URL https://icpmconference.org/2020/wp-content/uploads/sites/4/2020/10/ICPM_2020_paper_142.pdf.
- Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*, chapter 2, p. 68–125. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc.
- Leemans, S. J., Fahland, D., and Van Der Aalst, W. M. (2013). Discovering block-structured process models from event logs—a constructive approach. In *Application and Theory of Petri Nets and Concurrency: 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24–28, 2013. Proceedings 34*, p. 311–329. Springer.
- Levenshtein, V. I. et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, p. 707–710. Soviet Union.
- Neubauer, T. R. et al. (2022). Interactive trace clustering to enhance incident completion time prediction in process mining. In *CI4PM/PAI@ WCCI*, p. 8–20.
- Song, M., Gunther, W., Christian, and Van Der Aalst, W. M. (2008). Trace clustering in process mining. *International conference on business process management*, p. 109–120.
- Tariq, Z. et al. (2021). An event-level clustering framework for process mining using common sequential rules. In *Emerging Technologies in Computing: 4th EAI/IAER International Conference, iCETiC 2021, Virtual Event, August 18–19, 2021, Proceedings*, volume 4, p. 147–160. Springer International Publishing.
- Van Der Aalst, W. M. (2016). *Process mining: data science in action*, volume 2. Springer.
- Van Der Aalst, W. M. and Gunther, C. W. F. (2007). Finding structure in unstructured processes: The case for process mining. *Network analysis*, p. 3–12.
- Van Detten, J. N., Schumacher, P., and Leemans, S. J. (2023). An approximate inductive miner. In *2023 5th International Conference on Process Mining (ICPM)*, p. 129–136. IEEE.
- Van Dongen, B. (2012). Event log of a loan application process. *Eindhoven University of Technology*. URL https://data.4tu.nl/articles/BPI_Challenge_2012/12689204.
- Van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H., Weijters, A., and Van Der Aalst, W. M. (2005). The ProM framework: A new era in process mining tool support. In *International Conference on Application and Theory of Petri Nets*, p. 444–454. Springer.
- Wang, P. et al. (2018). A novel trace clustering technique based on constrained trace alignment. In *Human Centered Computing: Third International Conference, HCC 2017, Kazan, Russia, August 7–9, 2017, Revised Selected Papers*, volume 3, p. 53–63. Springer International Publishing.