Production Systems and Information Engineering
Volume 12 (2), pp. 23–39.
https://doi.org/10.32968/psaie.2024.2.2

EVALUATING PROCESS DISCOVERY FROM LOOP STRUCTURES

ERIKA BAKSÁNÉ VARGA
University of Miskolc, Hungary Institute of Information Technology
erika.b.varga@uni-miskolc.hu

ATTILA BAKSA
University of Miskolc, Hungary Faculty of Mechanical Engineering and Informatics
attila.baksa@uni-miskolc.hu

Abstract: Modern organizations increasingly rely on sophisticated information systems to manage their business processes, generating detailed event logs that record key activities. Process mining, and specifically process discovery, utilizes these event logs to construct models that represent the underlying processes. Effective process discovery is crucial for organizations to gain insights into their operations, identify inefficiencies, and drive continuous improvement. This study evaluates the capability of four process discovery algorithms – Alpha Miner, Heuristics Miner, ILP Miner, and Inductive Miner – in handling complex workflow patterns, particularly those involving intricate loop and nested control-flow structures. By generating synthetic event logs and applying the PM4Py library, we assess the algorithms' performance using metrics such as fitness, precision, generalization, and simplicity. Our results highlight the strengths and limitations of each algorithm, providing valuable insights for researchers and practitioners in the field.

Keywords: *process discovery, PM4Py, LOOP structures in workflows*

1. Introduction

Modern organizations increasingly rely on sophisticated information systems to manage their business processes, generating detailed event logs that record key activities. Process mining, and specifically process discovery, utilizes these event logs to construct models that represent the underlying processes [1]. Effective process discovery is crucial for organizations to gain insights into their operations, identify inefficiencies, and drive continuous improvement [2].

Process discovery algorithms automatically create process models from event logs, capturing the control-flow relationships between tasks [3]. These algorithms, such as Alpha Miner, Heuristics Miner, ILP Miner, and Inductive Miner, each strike different balances between model accuracy, complexity, and computational efficiency [4]. The Alpha Miner algorithm [5], one of the earliest and simplest, uses a basic set of rules to construct a Petri net representing the process. However, its simplicity often limits its applicability to more complex real-life logs. The Heuristics Miner [6] improves upon Alpha Miner by incorporating frequency-based heuristics, enabling it to better handle noise and infrequent behavior in the logs. ILP Miner [7] uses integer linear programming to discover models that guarantee soundness, however at a higher computational cost. Inductive Miner [8], known for its robustness, constructs process trees that can represent complex behaviors including parallelism, choices, and loops, making it particularly suitable for real-life event logs.

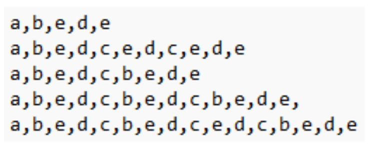
In this study, we focus on the complexities that arise in process models, particularly those involving intricate structures such as loops and nested control flows. We generate synthetic event logs from process models featuring these advanced constructs and evaluate the output of the aforementioned process discovery algorithms using the PM4Py library. By comparing the discovered models with the original models, we assess the algorithms' ability to accurately capture complex process behaviors. This research aims to highlight the strengths and limitations of current process discovery techniques when applied to sophisticated process structures, providing insights for both researchers and practitioners in the field.

2. Input for Process Discovery

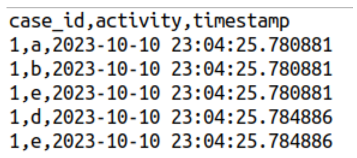
2.1. Build-up of Event Logs

Users can discover process models from event logs in various formats such as XES, MXML, CSV, OCEL, and others. These logs typically originate from ASCII text log files and contain information about process instances, such as their case ID, the activities involved and the timestamp.

In our study, synthetically generated TXT files contain sequences of events separated by commas, with each event specified by its symbolic name (a letter representing the activity). During the transformation process, the case ID and timestamp are automatically added as illustrated in *Figure 1*.

a) 

```
a,b,e,d,e
a,b,e,d,c,e,d,c,e,d,e
a,b,e,d,c,b,e,d,e
a,b,e,d,c,b,e,d,c,b,e,d,e,
a,b,e,d,c,b,e,d,c,e,d,c,b,e,d,e
```

b) 

```
case_id,activity,timestamp
1,a,2023-10-10 23:04:25.780881
1,b,2023-10-10 23:04:25.780881
1,e,2023-10-10 23:04:25.780881
1,d,2023-10-10 23:04:25.784886
1,e,2023-10-10 23:04:25.784886
```

Figure 1. a) Synthetic event sequences and
b) the first sequence in the transformed event log

The PM4Py library provides a uniform implementation for the examined process discovery algorithms, allowing each function to be called with the same arguments. Specifically, we provide the name of the input event log, the column names containing the case ID, activity/event label, and timestamp. The functions return descriptions of the generated process models, which can be visualized using GraphViz for graphical models and saved in PNG format. Outputs of algorithms producing declarative models can be stored in TXT format.

2.2. Complex Control Structures

Workflow patterns are essential for modeling and analyzing business processes, aiding in the comprehension and optimization of workflows. Traditional patterns include sequential, parallel, conditional, and loop structures, forming the foundation for most process models [9]. In this study, we focus on more complex control structures that go beyond these basic patterns (see *Figure 2*).

Loop structures allow for the repetition of one or more activities. An XOR-in-loop pattern combines exclusive choice (XOR) within a loop structure. Within the loop, at certain points, one of several possible paths is chosen based on a condition (XOR-split). After executing the selected path, the process returns to the beginning of the loop to repeat the embedded activities.

A loop-in-XOR pattern integrates loops within an exclusive choice (XOR) structure. Here, based on a condition, one of several paths is chosen at an XOR-split point, and the selected path contains a loop. The loop allows the repetition of activities within that particular path before returning to the XOR-split point.

A loop-in-loop pattern involves nested loops, where one loop is contained within another loop. The outer loop iterates a set of activities, and within those activities, there is another loop that repeats a subset of tasks.

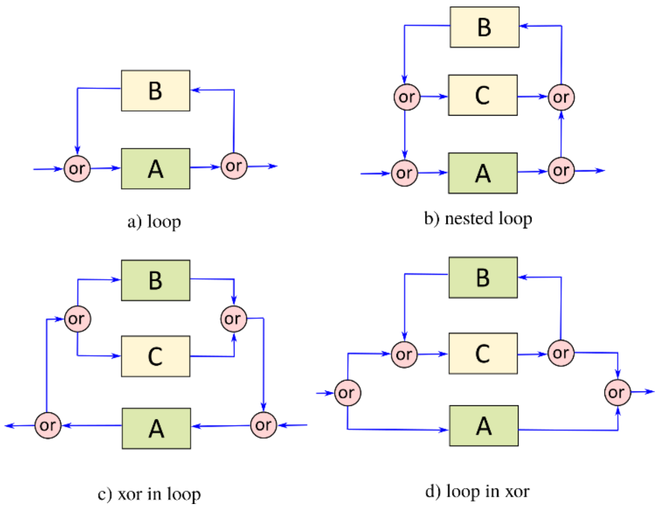


Figure 2. Complex control structures

For this research, we created process graphs containing these complex workflow patterns. By incorporating loops, XOR-in-loop, loop-in-XOR, and nested loops, we aim to evaluate the capability of process discovery algorithms in handling intricate process behaviors. Using these models, we generated synthetic event logs ensuring that all cases adhere to the predefined complex workflows [10]. These event logs serve as the basis for applying and assessing the performance of the PM4Py library's process discovery algorithms [11].

3. Methods

3.1. Process Discovery Algorithms

3.1.1. Alpha Miner

Alpha Miner is one of the earliest process mining algorithms designed to discover process models from event logs [5]. It identifies patterns in the sequences of activities recorded in the logs, specifically focusing on the order in which activities occur. The algorithm constructs a workflow net (a type of Petri net) that represents the control-flow structure of a business process as shown in Figure 3. Although Alpha Miner is foundational and simple, it has limitations such as difficulty handling noise and infrequent behavior, and it does not manage complex constructs like non-free-choice constructs well.

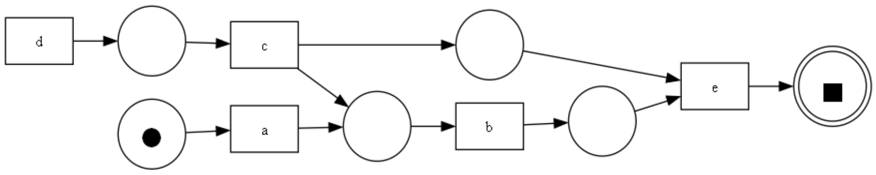


Figure 3. Process model generated by Alpha Miner from the even log in *Figure 1*

3.1.2. Heuristics Miner

Heuristics Miner [6] improves upon Alpha Miner by addressing some of its weaknesses. The main difference is that the Heuristic Miner applies filtering to reduce noise, meaning it removes insignificant or incomplete event log data to provide process models that are less precise but more fault-tolerant than those provided by the Alpha Miner. This algorithm uses dependency graphs called causal nets to represent the causal relations between activities (see *Figure 4*) and can handle loops and short-term dependencies more effectively. Heuristics Miner is particularly useful in practical scenarios where event logs are noisy and incomplete and contain large number of process cases.

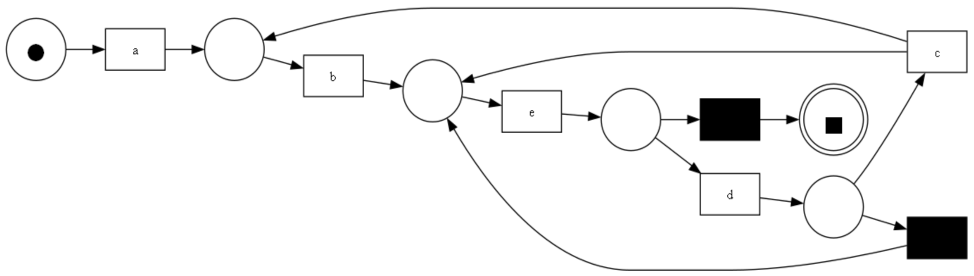


Figure 4. Petri net generated by Heuristics Miner from data in *Figure 1*

3.1.3. ILP Miner

The ILP (Integer Linear Programming) Miner [7] is an advanced process discovery algorithm that constructs process models by finding the best fitting model for the given event log using integer linear programming techniques. This algorithm aims to produce a Petri net that perfectly aligns with the observed behavior in the event log (see *Figure 5*) while ensuring certain formal properties, such as soundness and minimality. The ILP Miner is particularly effective in handling complex and large event logs, providing precise and comprehensive models that capture the complexities of the underlying processes. However, due to its computational intensity, it may be less suitable for very large datasets or real-time applications.

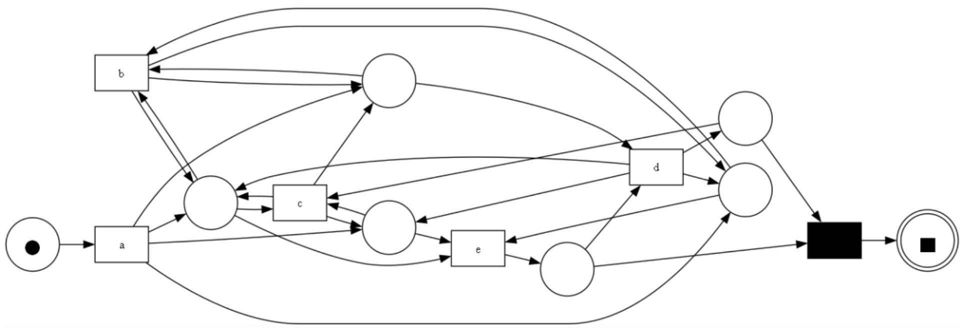


Figure 5. Process model generated by ILP Miner from the event log in Figure 1

3.1.4 Inductive Miner

Inductive Miner [8] represents a more advanced approach to process discovery. It builds process models using a divide-and-conquer strategy, ensuring the resultant models are block-structured (i.e., represented as process trees). This method guarantees sound process models that are easy to understand and manage. Inductive Miner can handle various complexities, including concurrency, and provides models that are easier to comprehend and modify. It is considered more versatile and reliable, especially for complex and large-scale processes.

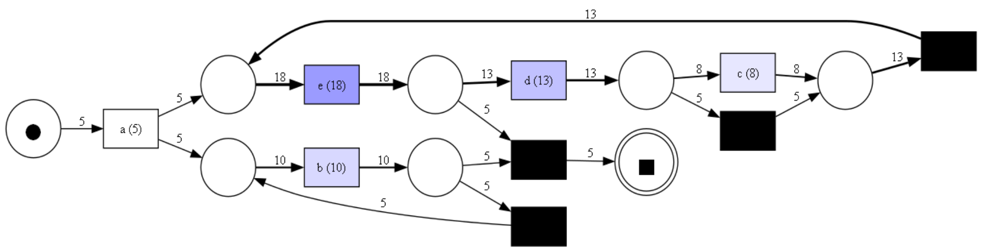


Figure 6. Petri net generated by Inductive Miner from the event log in Figure 1

3.2. Evaluation Metrics

In this study, we evaluate the models generated by the process discovery algorithms implemented in the PM4Py program library using the `pm4py.algo.evaluation` package, considering the following criteria:

- Fitness: This indicates the proportion of cases accurately captured by the model.
- Precision: It measures the percentage of the model-generated cases that are actually found in the original event log.
- Generalization: It evaluates the model's ability to generalize beyond the specific cases in the event log. A good generalization ensures the model can handle variations without overfitting to the log.
- Simplicity: It measures how straightforward and uncomplicated the discovered model is. A simpler model is preferred as it is easier to understand and interpret.

In addition to the above criteria, we also assess how well the discovered model aligns with the predefined process graph.

3.3. Test Data

We generated the test datasets described in Table 1 in TXT format using a graphical process graph editor and event log creator software [10], following the systematic structural layout outlined below:

- Process model containing single-depth XOR branches within a loop (LX) (3 files)
- Process model containing two XOR branches within a loop (L2X) (1 file)
- Process model containing two loops combined with a XOR branch (2LX) (1 file)
- Process model containing a loop embedded in a XOR branch (XL) (1 file)
- Process model containing a loop embedded in a XOR branch within a loop (LXL) (1 file)
- Process model containing two-level loop nesting (LL) (1 file)
- Process model containing two-level loop nesting within a XOR branch embedded in a loop (LXLL) (1 file)
- Process model containing single-depth XOR branch within four-level loop nesting (LLLLX) (1 file)

Table 1. Statistical data of the test datasets

Event log	No. of processes	Number of events	Number of event types
S_LX_01	1000	6576	4
S_LX_02	1000	8378	4
S_LX_05	1000	10922	8
S_L2X_01	1000	24725	9
S_2LX_02	1000	25469	10
S_XL_01	1000	4834	5
S_LXL_01	1000	23879	8
S_LL_02	1000	38167	5
S_LXLL_02	1000	50798	9
S_LLLLX_01	1000	237032	12

4. Results

4.1. XOR-in-loop

The log files containing single XOR-in-loop structures vary in complexity. The S_LX_01 dataset has a limited number of event types, with each activity occurring in a single structural position within the event sequence. In contrast, the S_LX_02 dataset includes event 'a' which can repeat in different structural positions within the sequence. The S_LX_05 log file features processes with multiple XOR branches within the loop, as shown in *Figure 7*.

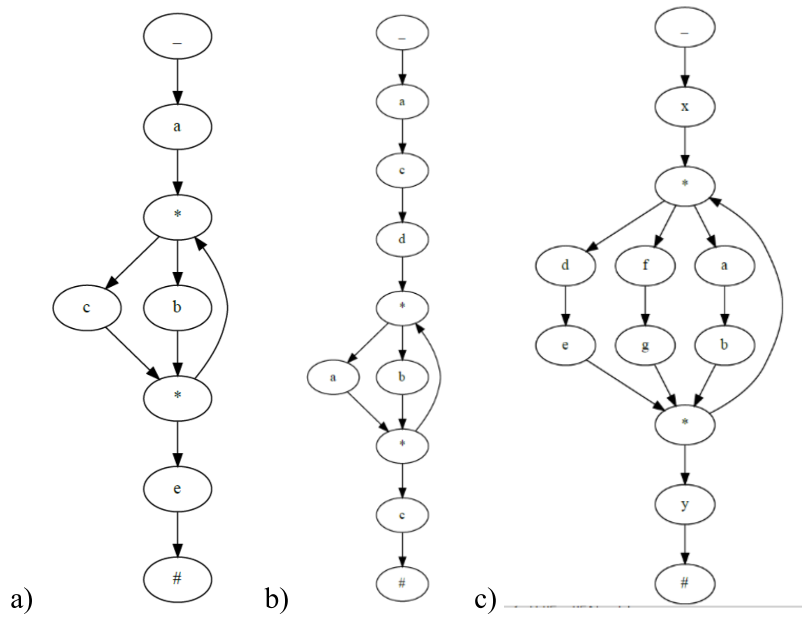


Figure 7. Original process models for the log files a) S_LX_01, b) S_LX_02, and c) S_LX_05

For the smaller datasets (S_LX_01 and S_LX_02), the Alpha Miner algorithm identifies the most frequent start and end activities but fails to detect the complex control-flow structures. Conversely, the ILP Miner method is effective only for these smaller datasets. Due to its limitations, particularly its inability to handle larger or more complex event logs effectively, we will not consider the ILP Miner method further in this study as it is not applicable for real-world event logs which typically exhibit greater complexity and variability.

Figures 8 and *9* show the process models generated by the Heuristics Miner and Inductive Miner algorithms for these datasets.

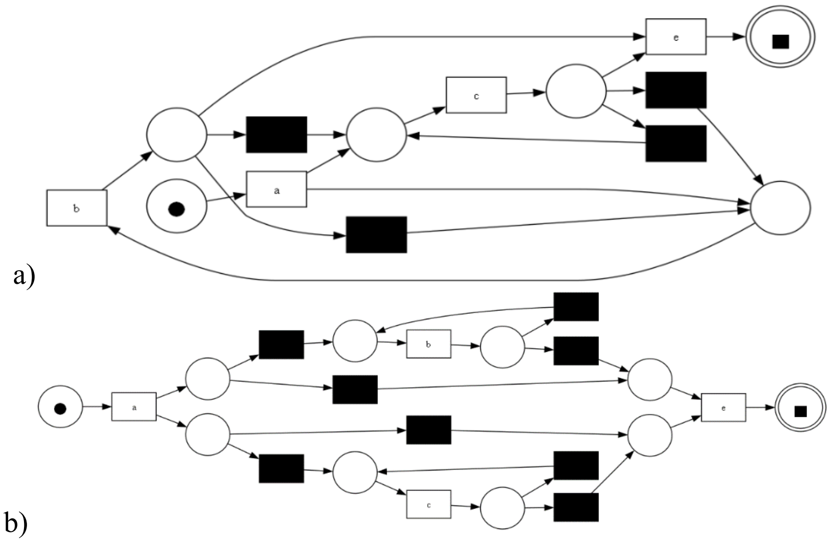


Figure 8. Process model for S_LX_01 dataset generated by a) Heuristics Miner and b) Inductive Miner

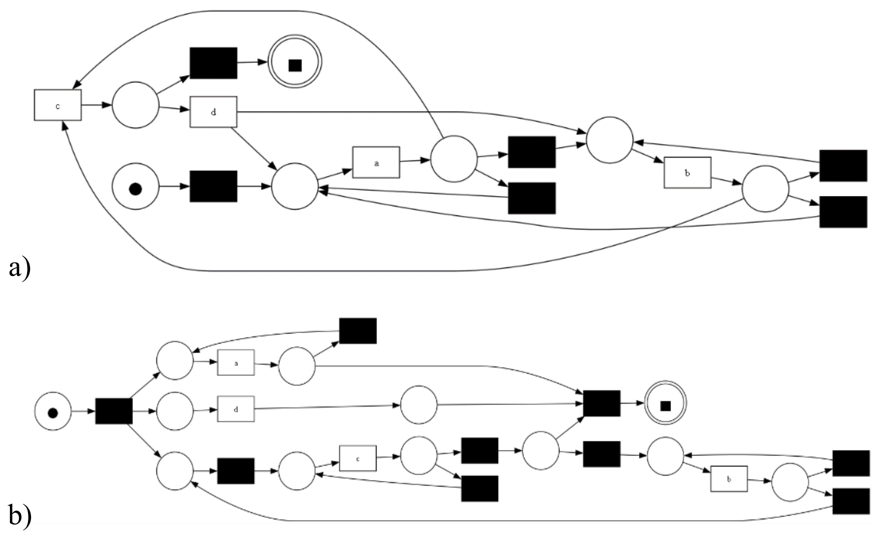


Figure 9. Process model for S_LX_02 dataset generated by a) Heuristics Miner and b) Inductive Miner

Figure 10 illustrates more complicated scenarios. The S_L2X_01 file represents two consecutive XOR branches within a loop, while the S_2LX_02 test file contains processes with two loops combined with a XOR branch.

When applying the Alpha Miner method to larger datasets (S_LX_05, S_L2X_01, and S_2LX_02), we observed that the method can handle XOR-in-loop structure with a limited number of possible options, as shown in *Figures 11* and *14*.

Figures 12, 13 and *14* display the process models generated by the Heuristics Miner and Inductive Miner algorithms for the S_L2X_01, S_LX_05 and S_2LX_02 datasets, respectively.

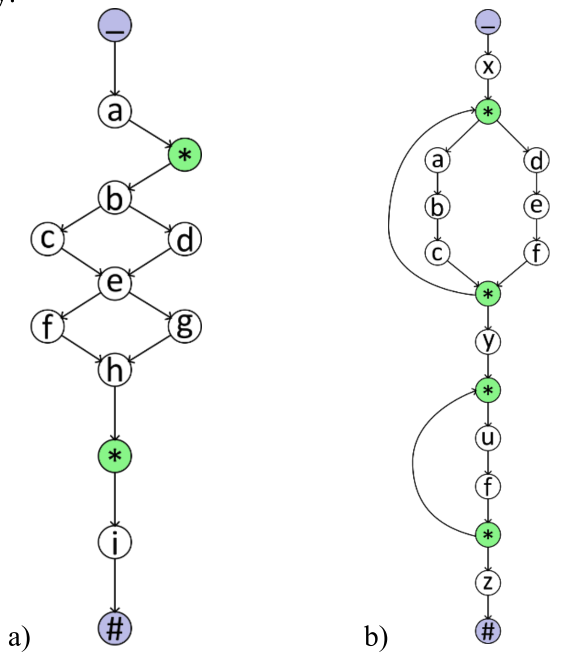


Figure 10. Original process graphs for the log files a) S_L2X_01, and b) S_2LX_02

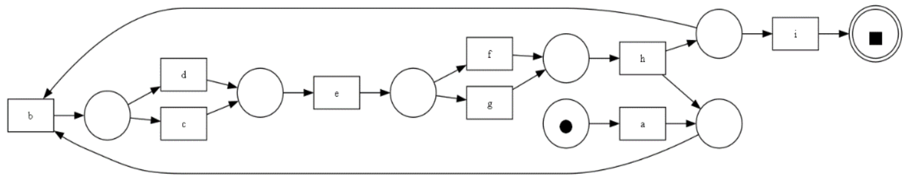


Figure 11. Process model for S_L2X_01 dataset generated by Alpha Miner

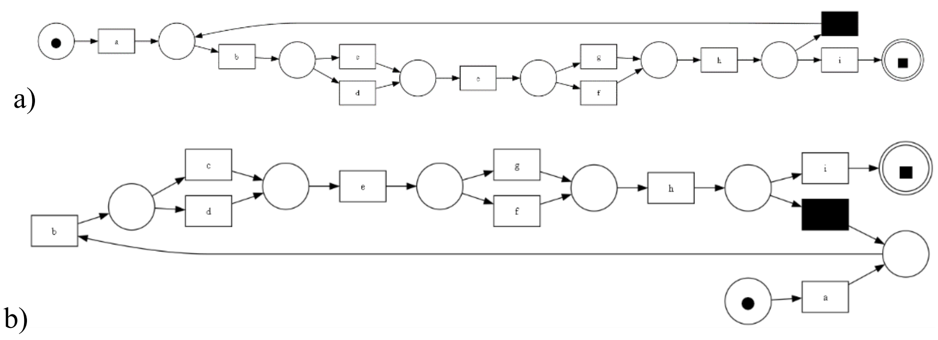
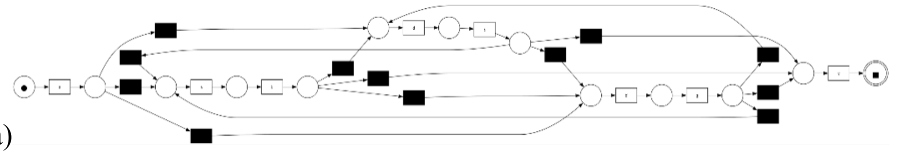
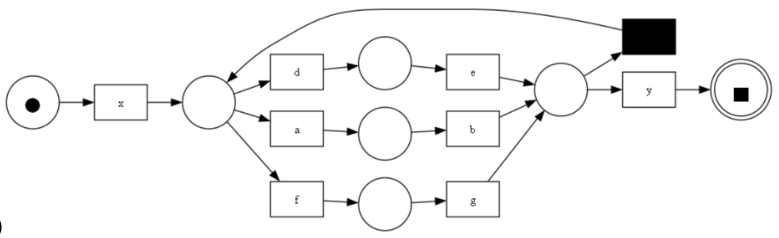


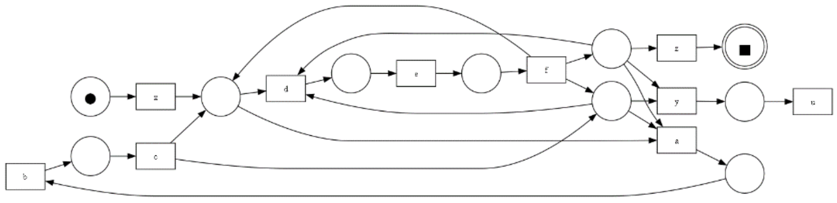
Figure 12. Process model for S_L2X_01 dataset generated by a) Heuristics Miner and b) Inductive Miner

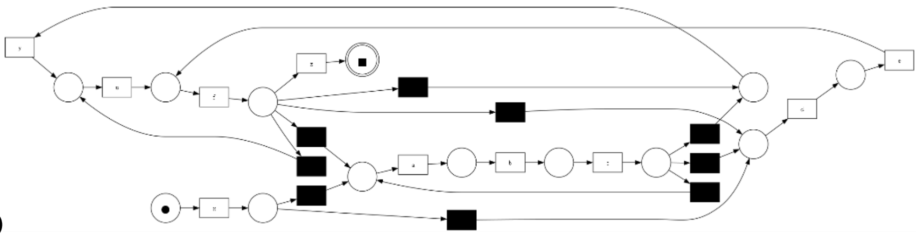

a)

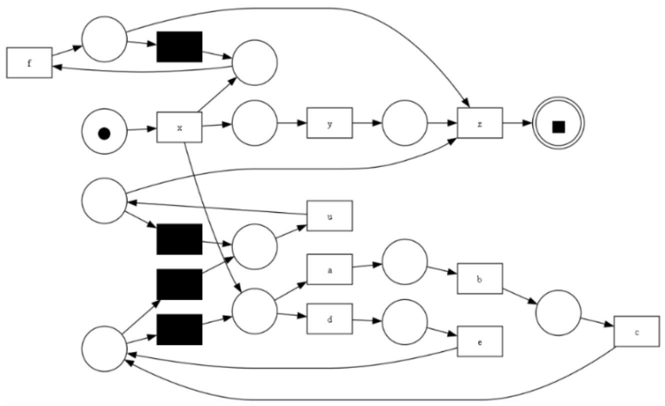


b)

Figure 13. Process model for S_LX_05 dataset generated by
a) Heuristics Miner and b) Inductive Miner


a)


b)



c)

Figure 14. Process model for S_2LX_02 dataset generated by
a) Alpha Miner, b) Heuristics Miner and c) Inductive Miner

4.2. Loop-in-XOR

In this category, the simplest dataset is S_XL_01 in terms of structure, whereas the S_LXL_01 log file contains a loop-in-XOR embedded in another loop. *Figure 15* demonstrates the process models of these event logs.

Studying the results, we can conclude that the Alpha Miner algorithm is unable to handle the Loop-in-XOR structure effectively. On the other hand, the Heuristics Miner and Inductive Miner algorithms produce the same output for the simpler S_XL_01 case, as shown in *Figure 16*. For the more complex S_LXL_01 dataset, the generated process models are presented in *Figure 17*.

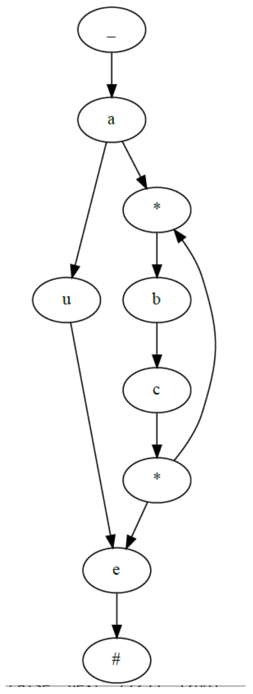
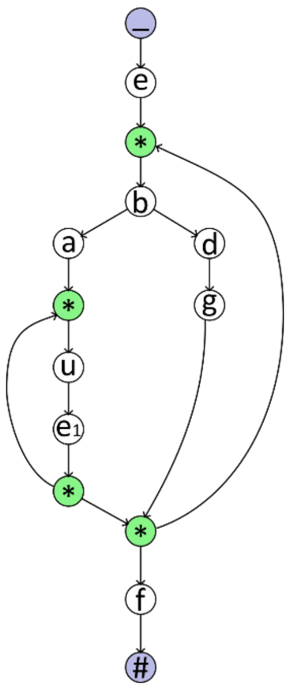
a)  b) 

Figure 15. Original process graphs for the log files a) S_XL_01, and b) S_LXL_01

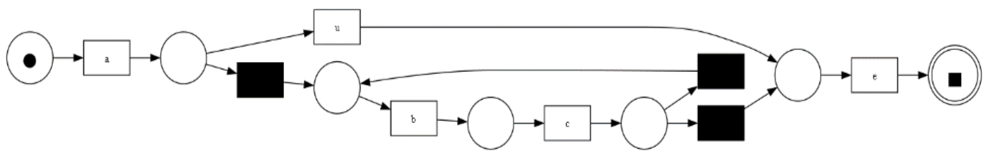


Figure 16. Process model for S_XL_01 dataset generated by Heuristics Miner and Inductive Miner

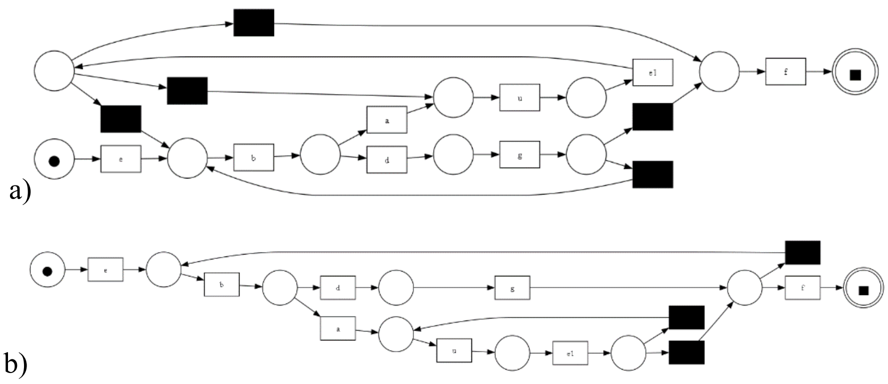


Figure 17. Process model for S_LXL_01 dataset generated by a) Heuristics Miner and b) Inductive Miner

4.3. Loop-in-loop

In the simplest approach, the loop-in-loop structure is not combined with XOR branches as shown in *Figure 18*. When executing the process discovery algorithms on the S_LL_02 dataset, we find that the Alpha Miner cannot capture loops if there are repetitive activities. In contrast, the Heuristics Miner can identify one-level loops, and the Inductive Miner can detect embedded loops. The generated models are displayed in *Figure 19*.

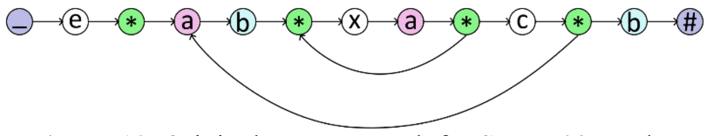


Figure 18. Original process graph for S_LL_02 test log

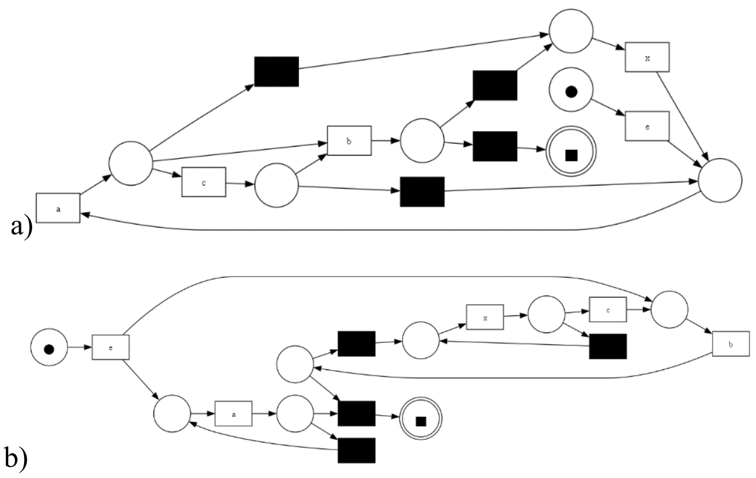


Figure 19. Process model for S_LL_02 dataset generated by a) Heuristics Miner and b) Inductive Miner

A more complex scenario arises when a loop-in-loop structure is within an XOR branch embedded in another loop (see *Figure 20a*). In this case, the Alpha Miner algorithm oversimplifies the model, failing to capture the composite control-flow patterns. The results of the Heuristics Miner and Inductive Miner methods for this structure are shown in *Figure 21*.

The most complicated case we created involves a process graph with four levels of loop embeddings combined with a XOR branch (see *Figure 20b*). For this scenario, the three examined algorithms – Alpha Miner, Heuristics Miner, and Inductive Miner – produce completely different models, as illustrated in *Figure 22*.

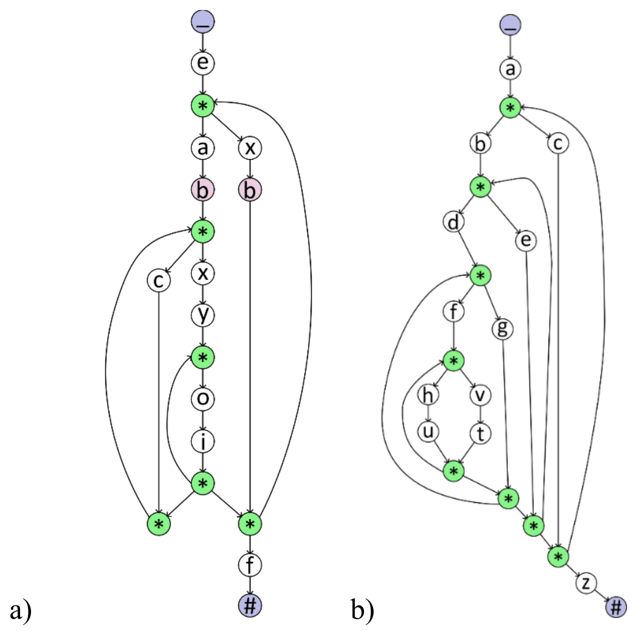


Figure 20. Original process graphs for the log files a) S_LXLL_02 and b) S_LLLLX_01

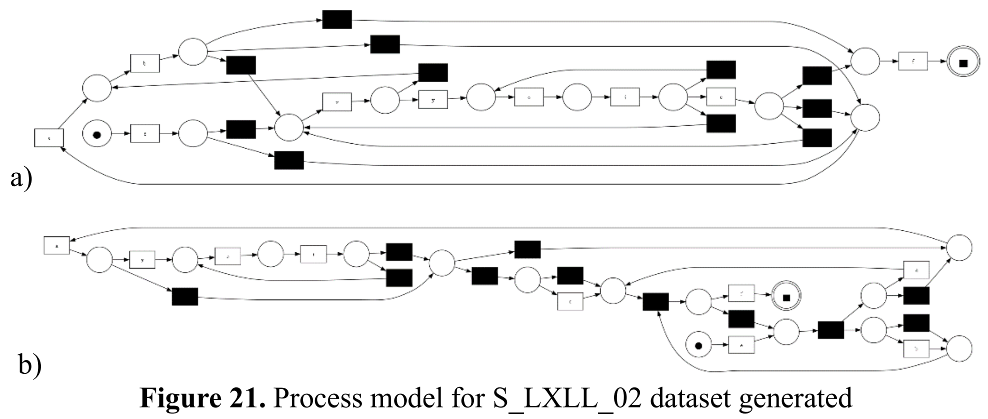


Figure 21. Process model for S_LXLL_02 dataset generated by a) Heuristics Miner and b) Inductive Miner

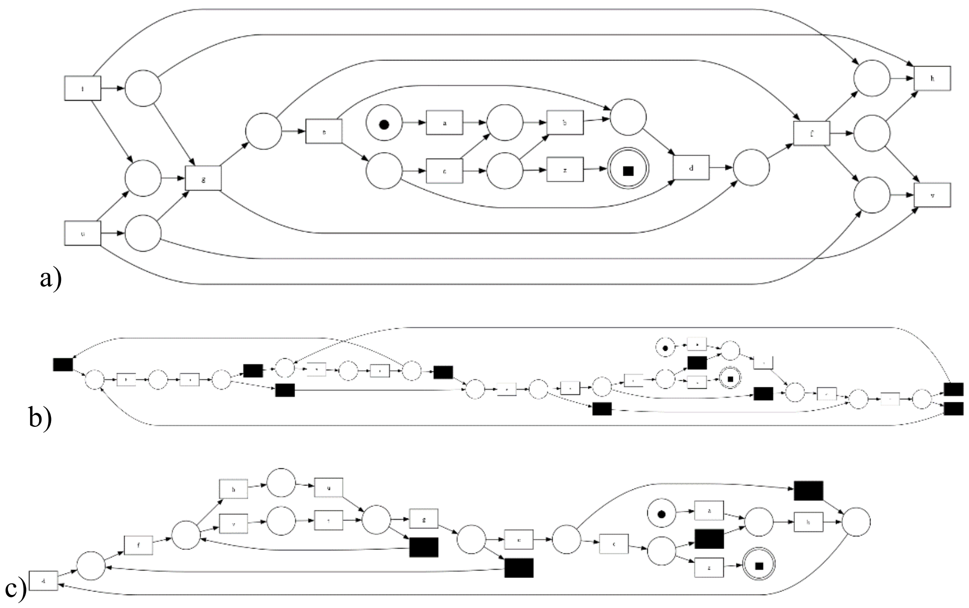


Figure 22. Process model for S_LLLLX_01 dataset generated by a) Alpha Miner, b) Heuristics Miner and c) Inductive Miner

4.4. Evaluation of Process Discovery Methods

The evaluation metrics for the process models generated by the examined algorithms are collected in Table 2. Some models are not sound and therefore cannot be replayed or aligned with the data in the event log. As a result, these models cannot be evaluated in terms of fitness, precision, generalization, and simplicity.

This evaluation highlights specific patterns for each algorithm. The Alpha Miner algorithm demonstrates a notable negative correlation between fitness and generalization ($r = -0.76$). This suggests that as the fitness of the model increases, its ability to generalize decreases. This trade-off indicates that Alpha Miner tends to overfit the models to the specific cases in the event log, limiting its applicability to broader scenarios. For this reason, this method produces a sound Petri-net process model only for 4 of the 10 synthetic event logs.

Heuristics Miner shows high precision and simplicity across various datasets. This algorithm is capable of producing models that are both accurate and relatively simple, making it suitable for datasets with varying complexities. It generated a replayable process model for 7 of the 10 test event logs. However, there is no significant correlation between the evaluation metrics, suggesting a balanced performance without any specific trade-offs.

ILP Miner consistently achieves high fitness across datasets, indicating that the discovered models align well with the recorded cases. However, this algorithm is effective mainly for smaller datasets and struggles with larger or more complex event logs. This limitation makes ILP Miner less practical for real-world applications where event logs are typically larger and more complex.

The Inductive Miner algorithm yields a significant negative correlation between precision and simplicity ($r = -0.81$). This implies that as the precision of the model increases, its simplicity tends to decrease. This trade-off suggests that achieving higher accuracy in representing the event log cases comes at the cost of creating more complex models. Inductive Miner produced 9 sound models out of 10, which proves its efficiency in detecting complex control-flow structures, including nested loops and XOR branches, making it a robust choice for handling intricate process models.

Table 2. Evaluation of the generated process models

Dataset	**Method**	**Fitness**	**Precision**	**Generalization**	**Simplicity**
S_LX_01	Alpha Miner	1.0	0.50	0.97	1.0
	Heuristics M.	0.97	0.97	0.97	0.63
	ILP Miner	1.0	0.93	0.97	0.66
	Inductive M.	1.0	0.93	0.96	0.73
S_LX_02	Alpha Miner	0.72	0.39	0.98	1.0
	Heuristics M.	–	–	–	–
	ILP Miner	1.0	0.69	0.97	0.46
	Inductive M.	1.0	0.52	0.97	0.71
S_LX_05	Alpha Miner	–	–	–	–
	Heuristics M.	0.84	0.98	0.96	0.70
	ILP Miner	1.0	0.35	0.97	1.0
	Inductive M.	1.0	0.91	0.97	0.80
S_L2X_01	Alpha Miner	–	–	–	–
	Heuristics M.	0.99	0.94	0.98	0.81
	ILP Miner	1.0	0.93	0.98	0.80
	Inductive M.	1.0	0.94	0.98	0.82
S_2LX_02	Alpha Miner	–	–	–	–
	Heuristics M.	0.99	0.75	0.97	0.72
	ILP Miner	1.0	0.93	0.98	0.37
	Inductive M.	1.0	0.54	0.98	0.73
S_XL_01	Alpha Miner	0.82	0.55	0.97	1.0
	Heuristics M.	0.99	0.99	0.96	0.88
	ILP Miner	1.0	0.50	0.97	0.60
	Inductive M.	1.0	0.99	0.96	0.88
S_LXL_01	Alpha Miner	–	–	–	–
	Heuristics M.	0.99	0.91	0.97	0.79
	ILP Miner	1.0	0.51	0.98	0.87
	Inductive M.	1.0	0.91	0.98	0.83
S_LL_02	Alpha Miner	0.23	0.49	0.98	1.0
	Heuristics M.	–	–	–	–
	ILP Miner	0.99	0.80	0.98	0.42
	Inductive M.	1.0	0.64	0.98	0.74
S_LXLL_02	Alpha Miner	–	–	–	–
	Heuristics M.	0.99	0.76	0.97	0.70
	ILP Miner	1.0	0.43	0.98	0.36
	Inductive M.	1.0	0.45	0.98	0.71
S_LLLLX_01	All methods	–	–	–	–

5. Conclusion

In this study, we evaluated the performance of four process discovery algorithms - Alpha Miner, Heuristics Miner, ILP Miner, and Inductive Miner – using synthetic event logs that represent complex workflow patterns. Our analysis focused on the algorithms' ability to accurately capture intricate control-flow structures, particularly those involving loops and nested loops combined with XOR branches. The key observations from our evaluation are as follows:

- Alpha Miner: The Alpha Miner algorithm, being one of the earliest and simplest, struggles with complex control-flow structures. It produced a sound Petri-net process model for only 4 of the 10 synthetic event logs. This highlights its limitations in handling noise, infrequent behavior, and complex constructs.
- Heuristics Miner: The Heuristics Miner algorithm improves upon Alpha Miner by incorporating frequency-based heuristics, making it more effective in practical scenarios with noisy and incomplete logs. This algorithm can produce models that are both accurate and relatively simple, making it suitable for datasets with varying complexities. It generated a sound Petri-net model for 7 of the 10 test event logs.
- ILP Miner: The ILP Miner algorithm, while precise and capable of handling complex event logs, is computationally intensive. Due to its high computational cost, it may be less suitable for very large datasets or real-time applications.
- Inductive Miner: The Inductive Miner algorithm proved to be the most versatile and reliable, particularly for complex and large-scale processes. It produced 9 sound Petri-net models out of 10, demonstrating its efficiency in detecting complex control-flow structures and providing models that are easy to understand and manage.

Our findings emphasize the importance of selecting the appropriate process discovery algorithm based on the complexity and characteristics of the event logs. While simpler algorithms like Alpha Miner may suffice for straightforward processes, more advanced methods like Inductive Miner are necessary for accurately capturing complex workflows. These insights can guide both researchers and practitioners in choosing the right tools for effective process discovery.

References

[1] Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A. (2013). Introduction to Business Process Management. In: *Fundamentals of Business Process Management.* Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33143-5_1.

[2] Lamghari, Z., Radgui, M., Saidi, R., and Rahmani, M. D. (2018). A set of indicators for BPM life cycle improvement. *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, Fez, Morocco, pp. 1–8, https://doi.org/10.1109/ISACV.2018.8354057.

[3] van der Aalst, W.M.P. (2016). Process Mining: Data Science in Action. 2nd ed. Springer. https://doi.org/10.1007/978-3-662-49851-4.

[4] Augusto, A. et al., (2019). Automated Discovery of Process Models from Event Logs: Review and Benchmark. *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705. https://doi.org/10.1109/TKDE.2018.2841877.

[5] van der Aalst, W. M. P., Weijters, A. J. M. M., and Maruster, L. (2004). Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16 (9), pp. 1128–1142. https://doi.org/10.1109/TKDE.2004.47

[6] Weijters, A. J. M. M., Ribeiro, J. T. S. (2011). Flexible Heuristics Miner (FHM). In Proceedings of the *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM),* pp. 310–317. https://doi.org/10.1109/CIDM.2011.5949453

[7] van der Werf, J. M. E. M., van Dongen, B. F. Hurkens, C. A. J., and Serebrenik, A. (2008). Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94 (3–4), pp. 387–412. https://doi.org/10.1007/978-3-540-68746-7_24

[8] Leemans, S. S. J., Fahland, D., and van der Aalst, W. M. P. (2013). Discovering Block-Structured Process Models from Event Logs – A Constructive Approach. In: *Application and Theory of Petri Nets and Concurrency*. Springer, pp. 311–329. https://doi.org/10.1007/978-3-642-38697-8_17.

[9] Russell, N., ter Hofstede, A. H. M., van der Aalst, W. M. P., and Mulyar, N. (2006). *Workflow Control-Flow Patterns: A Revised View*. BPM Center Report BPM-06-22, BPMcenter.org.

[10] Mileff, P. (2024). Design and Development of a Web-based Graph Editor and Simulator Application. *Production Systems and Information Engineering – ERPA Project*, Vol. 12 No. 2.

[11] Berti, A., van Zelst, S., Schuster, D. (2023). PM4Py: A process mining library for Python, *Software Impacts*, 17, 100556. https://doi.org/10.1016/j.simpa.2023.100556