PROJECT

Process Mining

Comparison between mining algorithms

Inductive Miner and Inductive Miner Directly Follows

MSc. in Computer Science &

Information Systems Engineering

Enrik Dollaj

Summary

Introduction

With the continued rise of Process Mining technology, more and more people are interested in learning more about this new area of data science and the details of how it works. Process mining algorithms are used to discover process models from event logs, which are detailed records of the activities that take place within a system or process. Process mining algorithms can be used to discover the underlying structure of a process, identify inefficiencies or bottlenecks, and monitor and improve the performance of the process over time.

This research paper delves into the comparative study of two advanced process mining algorithms: the Inductive Miner and its extension, the Inductive Miner Directly Follows. These algorithms combine business process discovery with conformance checking to provide users with an easy-to-use process mining exploration tool. The Inductive Miner is designed to discover process models from event logs in a structured and recursive manner. Meanwhile, the Inductive Miner directly Follows takes a more granular approach, it is a variant that focuses specifically on the directly follows relationship in the event logs.

The objectives of this project are as follows:

- Exploring the foundational theories behind each algorithm.
- Understanding the differences between the miners.
- Implementing these algorithms with a certain dataset using the pm4js library.
- Utilizing token-based conformance checking to thoroughly analyze and compare the results.

The document follows a step-by-step approach to compare the algorithms theoretically, describing the types of log files used, implementing them with pm4js and drawing insightful conclusions based on practical result comparisons. This comprehensive analysis aims to offer insights into process mining, helping organizations choose the most suitable tools and algorithms that suit their operational objectives.

Process Mining Algorithms

Inductive Miner

The Inductive Miner algorithm is a popular process discovery technique in process mining, known for *producing fitting and sound process models from event logs*. It employs a divide-and-conquer approach, starting with the entire log and recursively splitting it into smaller sub-logs based on identified patterns or "cuts." This recursive method ensures the construction of well-structured and interpretable models, typically represented as Petri nets. The algorithm can use the directly following relation to detect patterns and construct process blocks such as sequences, parallelisms, choices, and loops.

Steps of the Inductive Miner Algorithm:

1. *Base Case:* Create an empty model if the log is empty or contains no events.
2. *Finding Cut:* Identify a cut based on patterns like sequence, parallelism, or choice.
3. *Splitting Log:* Divide the log according to the identified cut.
4. *Recursion:* Apply the Inductive Miner algorithm recursively to each sub-log.
5. *Combining Results:* Integrate the process models of the subsets to form the final model.

The algorithm identifies various types of splits, such as sequential, parallel, concurrent, and loop, and uses these to decompose the event log into manageable parts. This decomposition continues until base cases are found. The resulting models use unique labels for visible transitions and hidden transitions specifically for loop splits, ultimately producing a Petri net or process tree that maps the process flow.

By focusing on structured and recursive analysis of directly follows relationships, the Inductive Miner ensures robust handling of real-world data, including noise, while delivering clear and precise process models.

Inductive Miner Directly Follows

The Inductive Miner - Directly Follows algorithm is a specialized variant of the Inductive Miner designed to leverage the directly follows relationship within event logs for efficient process discovery. This relationship captures the direct succession of activities, which is a fundamental concept in process discovery. The main feature of the algorithm is that it constructs a directly follows graph where nodes represent activities, and edges represent the directly follows relationship between activities. It uses simplified cuts based on the directly follows graph to split the event log. IMDF is often more efficient than the standard Inductive Miner due to its simplified approach and focus on directly following relationships.

Steps of the Inductive Miner - Directly Follows Algorithm:

1. *Directly Follows Graph Construction:* Construct the directly follows graph from the event log.
2. *Finding Cuts:* Identify cuts based on the directly follows graph, such as sequence cuts, parallel cuts, and loop cuts.
3. *Splitting Log:* Split the log according to the identified cuts.
4. *Recursion:* Recursively apply the IMDF algorithm to each subset of the log.
5. *Combining Results:* Combine the process models of the subsets to form the final model.

IMDF is particularly useful in scenarios where quick and efficient process discovery is required, such as real-time process monitoring and online process mining. The algorithm can adapt to various types of processes and event logs, ensuring robust performance across different domains. Its efficiency stems from its reliance on the directly follows relationship, which reduces the computational complexity typically associated with process discovery. This makes it an attractive option for organizations seeking to gain insights into their processes with minimal overhead, without sacrificing the quality and interpretability of the resulting models.

Comparison between Inductive Miner and Inductive Miner Directly Follows

The Inductive Miner and its variant, the Inductive Miner Directly Follows, are both prominent algorithms in process mining used to discover process models from event logs. However, they differ in several key aspects related to their approach, efficiency, and application.

- ## Approach

Inductive Miner: *Recursive and structured* means that it uses a divide-and-conquer approach, recursively splitting the event log based on identified patterns. And *detailed analysis* of the event log to detect various process blocks and splits.
Inductive Miner Directly Follows: *Simplified and efficient* because it focuses specifically on the directly follows relationship, constructing a directly follows graph to guide the discovery process.

- ## Directly Follows Relation

Inductive Miner: *Pattern identification* that utilizes the directly follows relationship but also incorporates other patterns and constructs process blocks beyond direct successions.
Inductive Miner Directly Follows: *Central focus* of the algorithm is the direct successions between activities.

- ## Model Output

Inductive Miner: Produces detailed process models in the form of *Petri nets*, which are mathematically rigorous and suitable for complex process analysis.
Inductive Miner Directly Follows: Typically generates process models that are less complex and easier to interpret, often represented as *process trees*.

- ## Noise Handling

Inductive Miner: *Robustness* which means that it has versions like Inductive Miner – Infrequent, designed to handle noisy data effectively by filtering out infrequent behaviors.

Inductive Miner Directly Follows: *Efficiency over robustness* so it may not handle noise as robustly as the standard Inductive Miner, as its primary design is for efficiency and simplicity.

- ## Efficiency

Inductive Miner: *Comprehensive* so may be less efficient due to the detailed analysis it performs on the event log.
Inductive Miner Directly Follows: *Faster* than the Inductive Miner due to its simplified approach and reduced computational complexity.

- ## Usage

Inductive Miner: *Complex Processes* which are suitable for discovering structured models from complex event logs, especially when noise handling is important.
Inductive Miner Directly Follows: *Real-time and Online Mining* means that is well-suited for real-time process monitoring and online process mining scenarios where quick insights are necessary.

Summary of Key Differences:

- ***Complexity vs. Simplicity:*** Inductive Miner is more comprehensive and suitable for complex, noisy logs, while IMDF is streamlined for efficiency and focuses on directly following relationships.
- ***Output Detail:*** Inductive Miner produces detailed Petri nets, whereas IMDF typically generates simpler process trees.
- ***Noise Handling:*** Inductive Miner handles noise better, whereas IMDF prioritizes speed over robustness to noise.
- ***Efficiency:*** IMDF is generally more efficient and faster due to its simplified approach.

In essence, the choice between Inductive Miner and IMDF depends on the specific requirements of the process mining task, such as the complexity of the event logs, the need for noise handling, and the necessity for efficiency and real-time analysis.

Log description

BPI Challenge 2020: Prepaid Travel Costs

This file contains the events related to prepaid travel costs Parent item: BPI Challenge 2020 The dataset contains events pertaining to two years of travel expense claims. In 2017, events were collected for two departments, in 2018 for the entire university. The various permits and declaration documents (domestic and international declarations, pre-paid travel costs and requests for payment) all follow a similar process flow. After submission by the employee, the request is sent for approval to the travel administration. If approved, the request is then forwarded to the budget owner and after that to the supervisor. If the budget owner and supervisor are the same person, then only one of these steps is taken. In some cases, the director also needs to approve the request.The process finished with either the trip taking place or a payment being requested and payed. On a high level, we distinguish two types of trips, namely domestic and international. For domestic trips, no prior permission is needed, i.e. an employee can undertake these trips and ask for reimbursement of the costs afterwards. For international trips, permission is needed from the supervisor. This permission is obtained by filing a travel-permit and this travel permit should be approved before making any arrangements. To get the costs for a trip reimbursed, a claim is filed. This can be done as soon as costs are actually paid (for example for flights or conference registration fees), or within two months after the trip (for example hotel and food costs which are usually paid on the spot).

Reference: 10.4121/uuid:5d2fe5e1-f91f-4a3b-ad9b-9e4126870165

JavaScript implementation of the algorithms

***GitLab**: https://github.com/EnrikFshn/InductiveMinerComparison.git*

- Importing the xes files with the pm4js XesImporter

```
// event log was imported
let eventLog = XesImporter.apply(data);
console.log('Imported event log:', eventLog);
```

```
require("pm4js/init");

const fs = require('fs');
const { XesImporter } = require('../node_modules/pm4js/pm4js/objects/log/importer/xes/importer.js');
const { InductiveMiner } = require('../node_modules/pm4js/pm4js/algo/discovery/inductive/algorithm.js');
const { FrequencyDfgDiscovery } = require('../node_modules/pm4js/pm4js/algo/discovery/dfg/algorithm.js');
const { ProcessTreeVanillaVisualizer } = require('../node_modules/pm4js/pm4js/visualization/process_tree/vanilla_graphviz.js');
const { ProcessTreeToPetriNetConverter } = require('../node_modules/pm4js/pm4js/objects/conversion/process_tree/to_petri_net.js');
const { PetriNetVanillaVisualizer } = require('../node_modules/pm4js/pm4js/visualization/petri_net/vanilla_graphviz.js');
const { WfNetToBpmnConverter } = require('../node_modules/pm4js/pm4js/objects/conversion/wf_net/to_bpmn.js');
const { DfgAlignments } = require('../node_modules/pm4js/pm4js/algo/conformance/alignments/dfg/algorithm.js');
const { TokenBasedReplay } = require('../node_modules/pm4js/pm4js/algo/conformance/tokenreplay/algorithm.js');
const { PetriNetAlignments } = require('../node_modules/pm4js/pm4js/algo/conformance/alignments/petri_net/algorithm.js');
const { TbrFitness } = require('../node_modules/pm4js/pm4js/algo/evaluation/petri_net/fitness/tbr.js');
const { AlignmentsFitness } = require('../node_modules/pm4js/pm4js/algo/evaluation/petri_net/fitness/alignments.js');
const { GeneralizationTbr } = require('../node_modules/pm4js/pm4js/algo/evaluation/petri_net/generalization/tbr_result.js');
const { SimplicityArcDegree } = require('../node_modules/pm4js/pm4js/algo/evaluation/petri_net/simplicity/arc_degree.js');
const { DfgSliders } = require('../node_modules/pm4js/pm4js/objects/dfg/util/sliders.js');
const { DfgPlayout } = require('../node_modules/pm4js/pm4js/algo/simulation/playout/dfg/algorithm.js');

fs.readFile('PrepaidTravelCost.xes', 'utf8', async (err, data) => {
    if (err) {
        console.error('Error reading file:', err);
        return;
    }
    console.log('File contents successfully read.');
```

Inductive Miner

1. Using the Inductive Miner to discover a process tree from the event log. Also there is the visualization of the process tree from the viz.js library.

```
// INDUCTIVE MINER of the event log gives a process tree
let processTree = InductiveMiner.applyPlugin(eventLog, "concept:name", 0.2);
console.log('Inductive Miner of the event log:', processTree);

// visualization of the process tree with the Viz.js library
let gv1 = ProcessTreeVanillaVisualizer.apply(processTree);
console.log('Process trees with inductive miner - Visualization vanilla Graphviz', gv1);
```

2. A process tree object can be converted to an accepting Petri net. Also there is the visualization of the Petri Net from the viz.js library.

```
// the process tree can be converted to an accepting Petri Net model
let acceptingPetriNet1 = ProcessTreeToPetriNetConverter.apply(processTree);
console.log('The accepting Petri Net of the event log from the Inductive Miner:', acceptingPetriNet1);

// visualization of the accepting Petri Net with the Viz.js library
let petriNetModel1 = PetriNetVanillaVisualizer.apply(acceptingPetriNet1);
console.log('Petri Net model with inductive miner - Visualization vanilla Graphviz', petriNetModel1);
```

3. Here is the conversion of Petri net to BPMN diagram. While Petri nets are of high utility in process mining, BPMN diagrams are more interesting for a business.

```
// converting an accepting Petri net to BPMN
let bpmnGraph1 = WfNetToBpmnConverter.apply(acceptingPetriNet1);
console.log('BPMN diagram of the Petri Net', bpmnGraph1);
```

4. Token-based replay is a popular conformance checking technique that measures conformance between event logs and accepting Petri nets.

```
// Token Based Replay
let tokenBasedReplayResult1 = TokenBasedReplay.apply(eventLog, acceptingPetriNet1);
console.log('Token based replay', tokenBasedReplayResult1);
```

5. The optimal alignments approach tries to find the best match between a process execution and a process model. The output of an alignment includes a list of moves, of which the first component is referring to the trace, and the second component is referring to the model, leading both the trace/process execution and the model from the initial to the final state. Here we consider alignments performed on accepting Petri nets.

```
// Alignments on Petri nets
let alignmentResultPN1 = PetriNetAlignments.apply(eventLog, acceptingPetriNet1)
console.log('Alignment on the Petri Nets', alignmentResultPN1);
```

6. The replay fitness on Petri nets can be computed on different dimensions: the percentage of the traces of the log which fit against the model, the

overall log fitness and the average of the fitness for the single traces of the log. The methods to calculate the fitness are derived from the implementations of token-based replay and alignments.

```
// Replay Fitness of Petri nets
let fitnessResultTbr1 = TbrFitness.apply(eventLog, acceptingPetriNet1);
console.log('Fitness of the event log against the accepting Petri net model using token-based replay.', fitnessResultTbr1);

let fitnessResultAlignment1 = AlignmentsFitness.apply(eventLog, acceptingPetriNet1);
console.log('Fitness of the event log against the accepting Petri net model using alignments.', fitnessResultAlignment1);
```

7. Generalization is a less-well-defined concept than fitness and precision, but it is still one of the four fundamental evaluation properties. It is defined as an average of the usage of the single transitions in the model during the replay. The more the transitions are needed during the replay of the traces of the log, the more the generalization is high.

```
// Generalization of Petri nets
let generalization1 = GeneralizationTbr.apply(eventLog, acceptingPetriNet1);
console.log('Generalization of Petri nets.', generalization1);
```

8. Simplicity is a less-well-defined concept than fitness and precision, but it is still one of the four fundamental evaluation properties. The average arc degree is calculated for both places and transitions. Then the simplicity is defined as an inverse of this mean degree.

```
// Simplicity of Petri nets
let simplicity1 = SimplicityArcDegree.apply(acceptingPetriNet1);
console.log('Simplicity of Petri nets.', simplicity1);
```

Inductive Miner Directly Follows

1. The Inductive Miner Directly-Follows is a version of the popular Inductive Miner algorithm that accepts as input a directly-follows graph and returns a process tree.

```
// INDUCTIVE MINER DIRECTLY FOLLOWS accepts as input a directly-follows graph and returns a process tree
let frequencyDfg = FrequencyDfgDiscovery.apply(eventLog, "concept:name");
let processTree1 = InductiveMiner.apply(null, null, 0.0, frequencyDfg);

console.log('Frequency Dfg of the event log:', frequencyDfg);
console.log('Inductive Miner Directly Follows of the event log:', processTree1);

// visualization of the process tree with the Viz.js library
let gv2 = ProcessTreeVanillaVisualizer.apply(processTree1);
console.log('Process trees with inductive miner directly follows - Visualization vanilla Graphviz', gv2);
```

2. The process tree can then be converted to a Petri net model, using the conversion method.

```
// the process tree can be converted to an accepting Petri Net model
let acceptingPetriNet2 = ProcessTreeToPetriNetConverter.apply(processTree1);
console.log('The accepting Petri Net of the event log from the Inductive Miner Directly Follows:', acceptingPetriNet2);

// visualization of the accepting Petri Net with the Viz.js library
let petriNetModel2 = PetriNetVanillaVisualizer.apply(acceptingPetriNet2);
console.log('Petri Net model with inductive miner directly follows - Visualization vanilla Graphviz', petriNetModel2);
```

3. Here is the conversion of Petri net to BPMN diagram.

```
// converting an accepting Petri net to BPMN
let bpmnGraph2 = WfNetToBpmnConverter.apply(acceptingPetriNet2);
console.log('BPMN diagram of the Petri Net', bpmnGraph2);
```

4. Here is the implementation of the token-based replay that measures conformance between event logs and accepting Petri nets.

```
// Token Based Replay
let tokenBasedReplayResult2 = TokenBasedReplay.apply(eventLog, acceptingPetriNet2);
console.log('Token based replay', tokenBasedReplayResult2);
```

5. The optimal alignments approach on DFGs works in a similar way to the approach on Petri nets.

```
// Alignments on Directly Follows Graphs
let alignmentResultDFG = DfgAlignments.apply(eventLog, frequencyDfg);
console.log('Alignment on the DFG', alignmentResultDFG);
```

6. The optimal alignments approach performed on accepting Petri nets.

```
// Alignments on Petri nets
let alignmentResultPN2 = PetriNetAlignments.apply(eventLog, acceptingPetriNet2)
console.log('Alignment on the Petri Nets', alignmentResultPN2);
```

7. The replay fitness on Petri nets computed on 2 different dimensions: the fitness of the event log against the accepting Petri net model using token-based replay and the fitness of the event log against the accepting Petri net model using alignments.

```
// Replay Fitness of Petri nets
let fitnessResultTbr2 = TbrFitness.apply(eventLog, acceptingPetriNet2)
console.log('Fitness of the event log against the accepting Petri net model using token-based replay.', fitnessResultTbr2);

let fitnessResultAlignment2 = AlignmentsFitness.apply(eventLog, acceptingPetriNet2)
console.log('Fitness of the event log against the accepting Petri net model using alignments.', fitnessResultAlignment2);
```

8. Generalization of the Petri Net.

```
// Generalization of Petri nets
let generalization2 = GeneralizationTbr.apply(eventLog, acceptingPetriNet2);
console.log('Generalization of Petri nets.', generalization2);
```

9. Simplicity of the Petri Net.

```
// Simplicity of Petri nets
let simplicity2 = SimplicityArcDegree.apply(acceptingPetriNet2);
console.log('Simplicity of Petri nets.', simplicity2);
```

10. PM4JS offers the possibility to slide the directly-follows graph, restricting the behavior to the desired number of activities/paths.To filter the directly-follows graph on the specified percentage of activities the following command can be used.

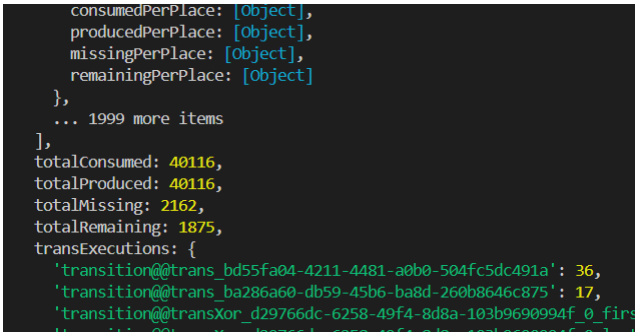
```
// Sliding Directly Follows Graphs
let filteredDfg = DfgSliders.filterDfgOnPercActivities(frequencyDfg, 0.2);
console.log('Sliding Directly Follows Graphs', filteredDfg);
```

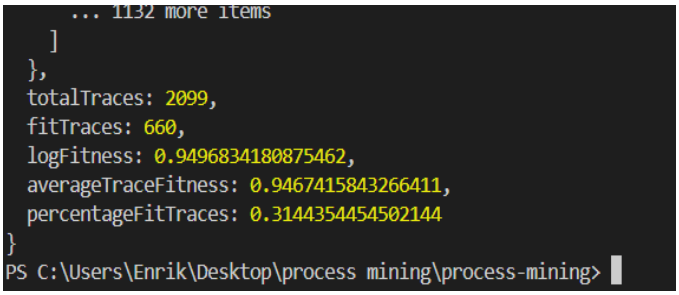
11. A playout operation returns an event log with a set of traces that are allowed by the directly-followed graph. To execute a playout of a DFG, and get the simulated log, the following command can be executed

```
// Playout of a DFG
let simulatedLog = DfgPlayout.apply(frequencyDfg);
console.log('Playout of a DFG', simulatedLog);
```

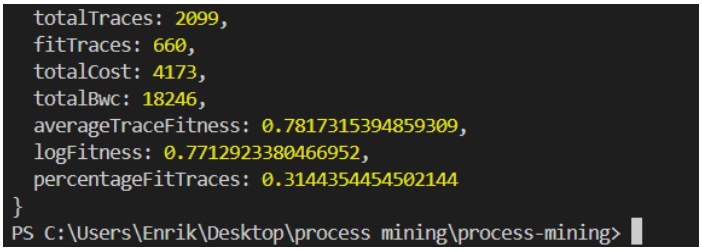
Presenting results

Inductive Miner algorithm: Token based replay fitness

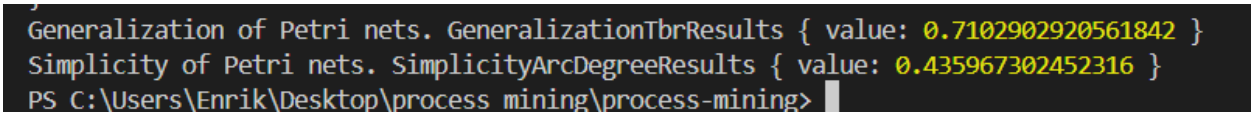




Inductive Miner algorithm: Alignments on Petri Nets



Inductive Miner algorithm: Generalization and Simplicity of the Petri Nets



Inductive Miner Directly Follows algorithm: Token based replay fitness

```
  totalTraces: 2099,
  fitTraces: 0,
  logFitness: 0.8276339565292017,
  averageTraceFitness: 0.827393784963945,
  percentageFitTraces: 0
}
PS C:\Users\Enrik\Desktop\process mining\process-mining>
```

Inductive Miner Directly Follows algorithm: Alignments on Directly Follows Graph

```
  },
  totalTraces: 2099,
  fitTraces: 2099,
  totalCost: 0,
  totalBwc: 20345,
  averageTraceFitness: 1,
  logFitness: 1,
  percentageFitTraces: 1
}
```

Inductive Miner Directly Follows algorithm: Generalization and Simplicity of the Petri Nets

```
Generalization of Petri nets. GeneralizationTbrResults { value: 0.608964117247415 }
Simplicity of Petri nets. SimplicityArcDegreeResults { value: 0.425273390036452 }
PS C:\Users\Enrik\Desktop\process mining\process-mining>
```

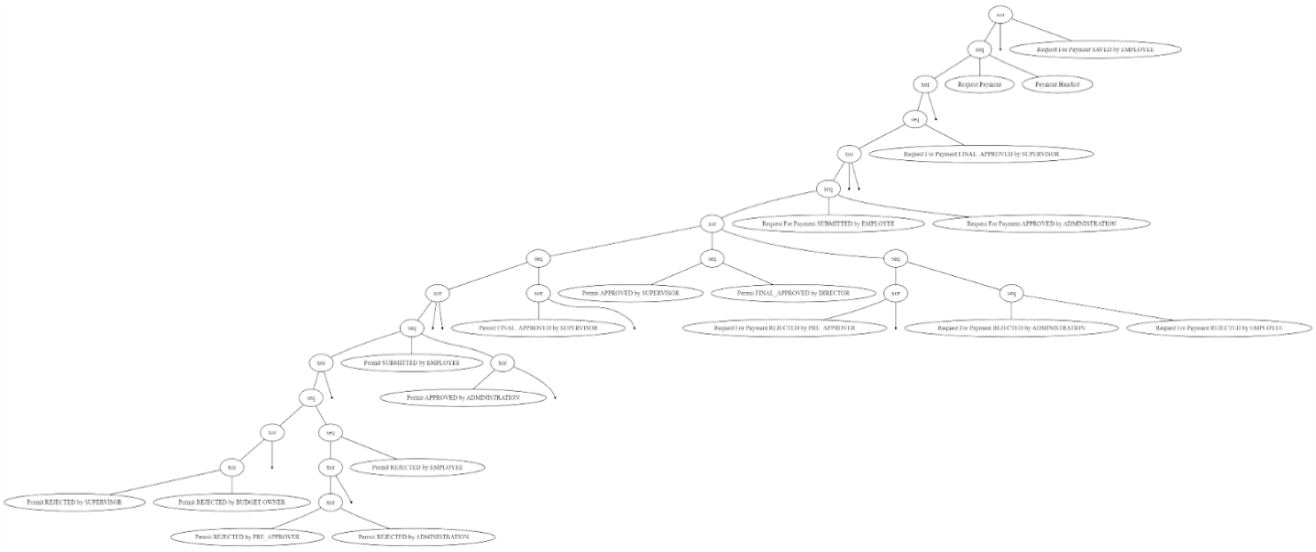
Inductive Miner Directly Follows algorithm: sliding DFG with a percentage of 0.2 out of 1 of the number of activities to be shown

```
Sliding Directly Follows Graphs FrequencyDfg {
  activities: {
    'Permit SUBMITTED by EMPLOYEE': 1952,
    'Request For Payment SUBMITTED by EMPLOYEE': 2279,
    'Request For Payment FINAL_APPROVED by SUPERVISOR': 1958,
    'Request Payment': 1990,
    'Payment Handled': 1990,
    'Request For Payment APPROVED by ADMINISTRATION': 1726,
    'Permit APPROVED by ADMINISTRATION': 1603
  },
  startActivities: {
    'Permit SUBMITTED by EMPLOYEE': 1859,
    'Request For Payment SUBMITTED by EMPLOYEE': 233
  },
  endActivities: { 'Payment Handled': 1970, 'Request Payment': 1 },
  pathsFrequency: {
    'Request For Payment SUBMITTED by EMPLOYEE,Request For Payment FINAL_APPROVED by SUPERVISOR': 135,
    'Request For Payment FINAL_APPROVED by SUPERVISOR,Request Payment': 1905,
    'Request Payment,Payment Handled': 1969,
    'Payment Handled,Permit SUBMITTED by EMPLOYEE': 1,
    'Request For Payment SUBMITTED by EMPLOYEE,Request For Payment APPROVED by ADMINISTRATION': 1718,
    'Request For Payment APPROVED by ADMINISTRATION,Request For Payment FINAL_APPROVED by SUPERVISOR': 931,
    'Permit SUBMITTED by EMPLOYEE,Permit APPROVED by ADMINISTRATION': 1603,
    'Permit APPROVED by ADMINISTRATION,Request For Payment SUBMITTED by EMPLOYEE': 163,
    'Permit APPROVED by ADMINISTRATION,Request For Payment APPROVED by ADMINISTRATION': 1,
    'Permit APPROVED by ADMINISTRATION,Payment Handled': 1,
    'Permit APPROVED by ADMINISTRATION,Request Payment': 1,
    'Payment Handled,Request Payment': 1,
    'Request For Payment FINAL_APPROVED by SUPERVISOR,Payment Handled': 1
  }
}
PS C:\Users\Enrik\Desktop\process mining\process-mining>
```

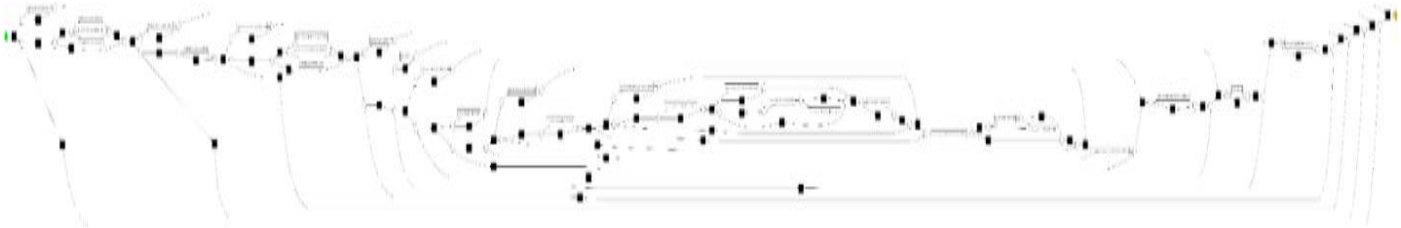
Additional Information about the results in the pictures above:

- **totalConsumed**: the number of consumed tokens among all the replayed cases.
- **totalProduced**: the number of produced tokens among all the replayed cases.
- **totalMissing**: the number of missing tokens among all the replayed cases.
- **totalRemaining**: the number of remaining tokens among all the replayed cases.
- **totalTraces**: the number of different cases in the replayed log.
- **fitTraces**: the number of cases that are perfectly fit according to the Petri net model.
- **logFitness**: the log fitness 0.5(1 - M/C) + 0.5(1 - R/P). * result: contains a dictionary (replay results) for every case of the log.
- **averageTraceFitness***: the average of the fitness at the trace level, for all the cases of the log.*
- **percentageFitTraces**: the ratio (between 0 and 1) of the traces of the log which fit against the model.
- **totalCost**: the sum of the costs of all the alignments that are performed.
- **totalBwc**: the sum of the length of the trace and the length of the shortest path in the model taking from the initial marking to the final marking.

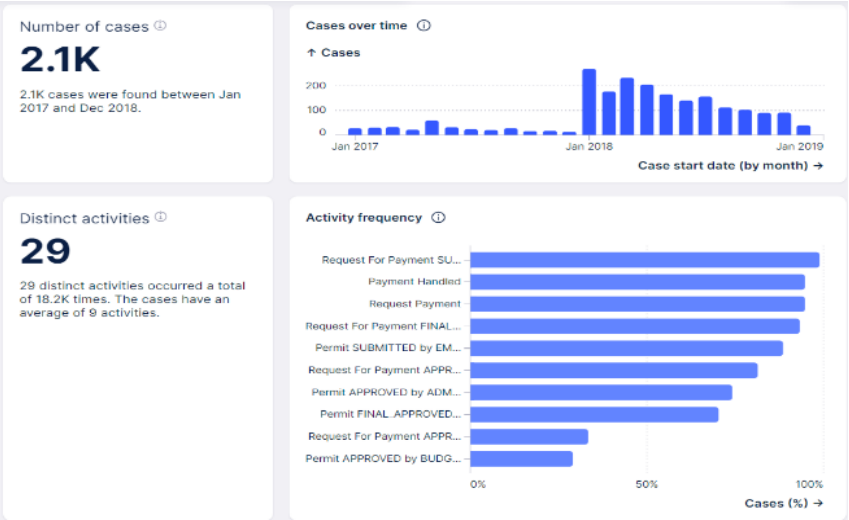
Inductive Miner



Inductive Miner Directly Follows



Celonis tool:



Result analysis and conclusions

Conformance checking and fitness comparison are crucial evaluation techniques in process mining, used to assess how well discovered process models represent the actual behavior recorded in event logs. The comparison between the Inductive Miner and the Inductive Miner Directly Follows offers valuable insights into their performance and suitability for process mining tasks.

Analysis of Conformance Checking & Evaluation results

Inductive Miner: Achieved a log fitness score of approximately 0.94, indicating a strong conformance between the process model generated by the algorithm and the actual event log data. This suggests that the algorithm can effectively replay the event log traces accurately. Also the average trace fitness that represents the fitness score at the trace level, is basically the same value as the log fitness. Even the generalization and simplicity scores were higher than the algorithm's variant, 0.71 and 0.43 respectively, which assess the model's ability to generalize from the event log and show how simple and understandable the model is.

Inductive Miner Directly Follows: Demonstrated a slightly lower fitness score of approximately 0.827, indicating that it still is a good conformance score. Like in the Inductive Miner, the average trace fitness is approximately the same as the log score. While the score for the generalization was 0.608 and for the simplicity 0.42.

Interpreting the results

The higher fitness score from the event log proved the capabilities of the Inductive Miner to give an accurate process model. While both algorithms performed well in capturing the overall process flow, the Inductive Miner highlights its tendency to provide precise information due to its detailed analysis and noise-handling capabilities. But because of its complexity, maybe the IMDF might be still a choice to consider because it offers efficient and simpler models with good fitness and high precision, as we saw in its fitness scores and the simplicity score which were nearly the same between the two.

Conclusion

Overall, the choice between Inductive Miner and IMDF depends on the specific needs: Inductive Miner for more detailed and robust models in complex environments, and IMDF for simpler, efficient models in scenarios where quick insights are needed. Both algorithms are widely used in process mining tools and research for discovering accurate process models from event logs.
With process mining being adopted at such a rapid pace, the technology itself continues to advance at an incredible rate. With this, the mining algorithms will also mature as Process Mining providers explore new ways to best represent the process data and maximize a business's ability to gain insight into how their processes actually work.

References

Mindzie:

https://mindzie.com/2022/12/15/types-of-process-mining-algorithms/#:~:text=Like%20the%20Alpha%20Miner%20algorithm,models%20from%20noisy%20event%20logs.

AIMultiple Research:

https://research.aimultiple.com/process-mining-algorithms/

Inductive visual Miner manual by Sander J.J. Leemans:

https://www.leemans.ch/publications/ivm.pdf

Inductive visual Miner & Directly Follows visual Miner manual by Sander J.J. Leemans:

https://leemans.ch/publications/ivm%20ProM%206.9.pdf

Dataset:

10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972

Documentation of the pm4js library for the implementation of the code:

https://www.npmjs.com/package/pm4js?activeTab=readme

Visualization of the petri net models:

https://viz-js.com/

Celonis:

https://academic-celonis-7dl4x5.eu-2.celonis.cloud/process-workspace/ui/assets/5835ec5c-585d-45af-bc69-673ac89335fa

Tools

1. Chrome browser for retrieving information
2. Javascript programming language for implementing the analysis
3. Visual Studio Code for the development environment to create the script
4. GitLab Version Control