Introduction to process discovery (Directly Follow Graph)

Jakub Kot, Piotr Kubala, Rafał Łukosz, Piotr Karaś, Tomasz Kawiak

October 31, 2025

1 Task 1

- *How many events are in the log?*
  There are 9942 events in the log, based on the sum of edge labels in the final graph, which represents the total number of transitions between activities.

- *Which task (Activity) takes the longest time? (Of all the tasks that took the longest time in any process instance)*
  The task that takes the longest time is 'Repair (Complex)' with a maximum duration of 0 days 00:50:00. We found the task using the following code.

```python
df['Start Timestamp'] = pd.to_datetime(df['Start Timestamp'])
df['Complete Timestamp'] = pd.to_datetime(df['Complete Timestamp'])

df['duration'] = df['Complete Timestamp'] - df['Start Timestamp']
longest_duration_per_activity = df.groupby('Activity')['duration'].max()

longest_task = longest_duration_per_activity.idxmax()
longest_duration = longest_duration_per_activity.max()

print(f"The task that takes the longest time is '{longest_task}' with a
    maximum duration of {longest_duration}.")
```

- *What conclusions can be drawn from the discovered process of phone repair?*

  Based on the event log and the process model:

  - The discovered phone-repair process contains 9942 events, which indicates a reasonably large dataset and nontrivial process variability.
  - From the discovered model we observe that the dominant path is Register → Analyze Defect → Repair (Complex) → Inform User → Test Repair → Archive Repair → End, which covers approximately 23% of cases (253 out of 1104).
  - The activity 'Repair (Complex)' has the largest maximum duration and appears to be a significant bottleneck. Fixing throughput here would likely reduce overall lead time.
  - There are several short loops between activities such as between 'Test Repair' and 'Restart Repair', suggesting frequent rework or checks which could increase cost and cycle time.
  - Resource analysis shows 6 distinct Testers and 6 distinct Solvers, indicating the task is split across multiple people, which might lead to possible coordination overhead.

  Model suggests two improvement directions:

– Reduce rework by clarifying acceptance criteria and consolidating inspection steps,
– Address the bottleneck activity (e.g., by staffing changes, process redesign or automation).

2 Task 2

Modify the task labels in the model so that for each task, in addition to the name, the number of occurrences of the corresponding log event is also displayed.

We found out that ‘ev_counter[event]‘ is responsible for counting the occurrences of each event in the log, so we simply added that to the label of the node.

The result is presented in the image 1.

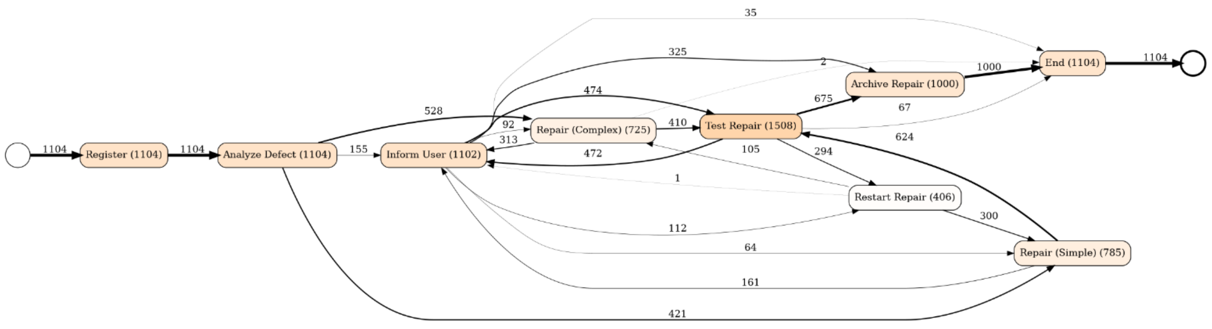


Figure 1: Implementation of the number of occurences in the node labels.

3 Task 3, 4 and 5

Currently the process displays the total number of occurrences. Provide the possibility to change it in such a way that a user can choose: Absolute frequency (this is the option already provided), Case frequency (unique cases that contain event or flow), Max repetitions, Case coverage (percentage of cases that include event or flow).
Add filtering option (by events or flows) according to certain thresholds to show or hide tasks or flows according to the selected threshold. This might be very naive method of setting specific thresholds for the events or flows.
Add the possibility for the user to set thresholds (by using a slider with ‘ipywidgets‘, possibly specifying a number or percentage, based what a user wants) and display the filtered model when the thresholds are changed. Test the functionality on different thresholds to determine if the model looks correct, in particular if any tasks become detached from the model or are not properly linked to the model. Check the behavior for selected thresholds for frequency, e.g. if you filter by absolute frequency (total number of occurrences) you can check the following numbers: 420 for flows, 700 for events (separately and together), 500 for flows and events, 0 and 1000 for flows or events. Determine automatically what should be the numbers for 0% and 100%. Provide the appropriate screenshots in the lab report.

Both the possiblity to change the way the number of occurences is displayed and the filtering option for both events and flows were implemented. The filtering is done using a naive method of simply getting rid of any events/flows that have the metric smaller than the set threshold.

The filtering process was wrapped using ipywidgets to allow the user to select the aggregation option, filtering mode and thresholds easily and dynamically view the results.

The changes are too big to show present them as code snippets, so the full code has been attached to the report.

The provided behaviors were analyzed. However, due to the naive implementation, most of the filtration results are simply disconnected graphs.

The final result, as well as the different behaviors are presented in the pictures below.

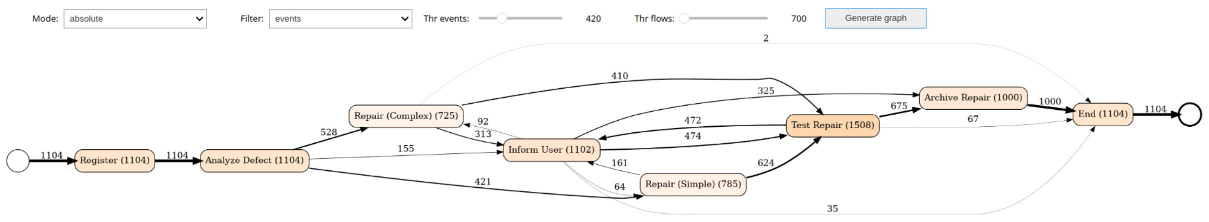

Mode: absolute Filter: events Thr events: 420 Thr flows: 700

Figure 2: Flows $\geq$ 420

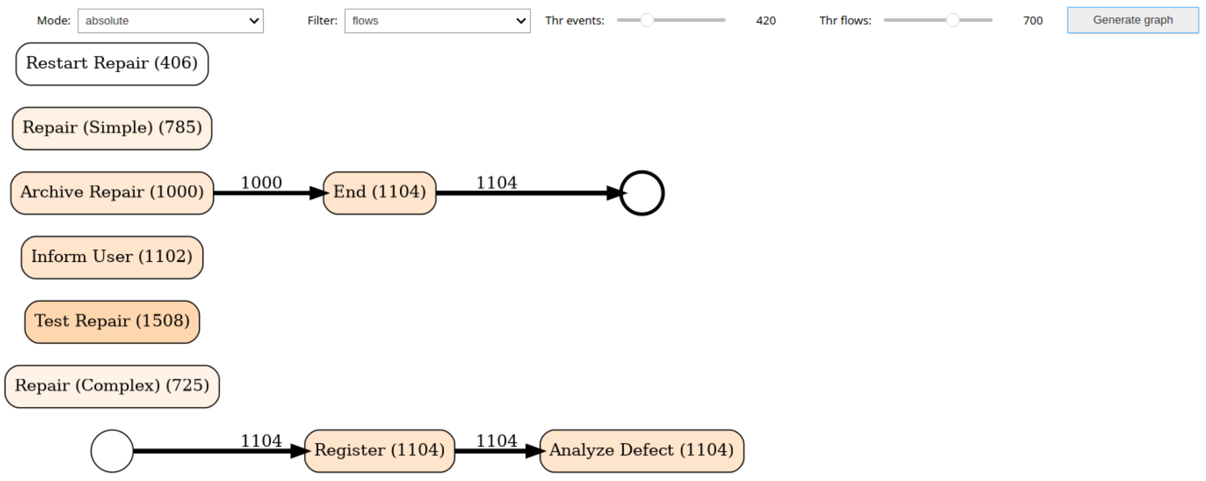

Mode: absolute Filter: flows Thr events: 420 Thr flows: 700

Figure 3: Events $\geq$ 700

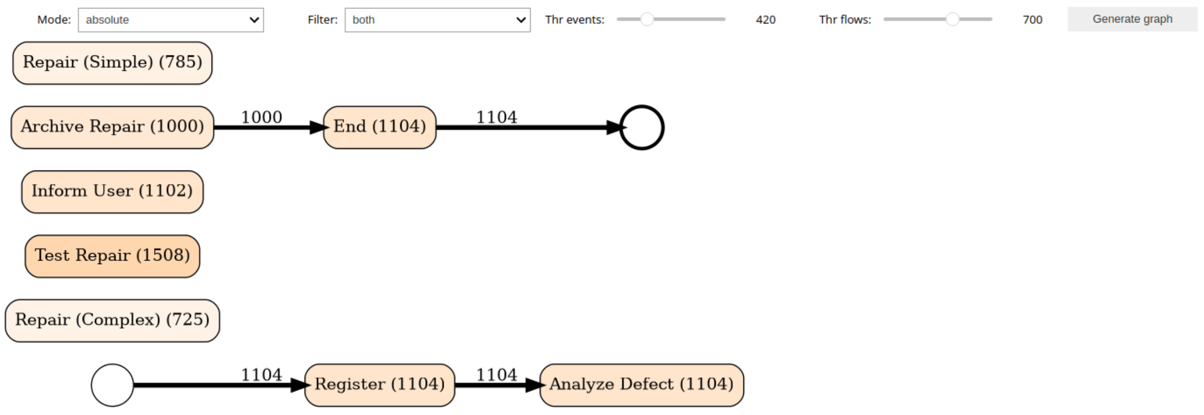

Mode: absolute Filter: both Thr events: 420 Thr flows: 700

Figure 4: Flows $\geq$ 420 and Events $\geq$ 700

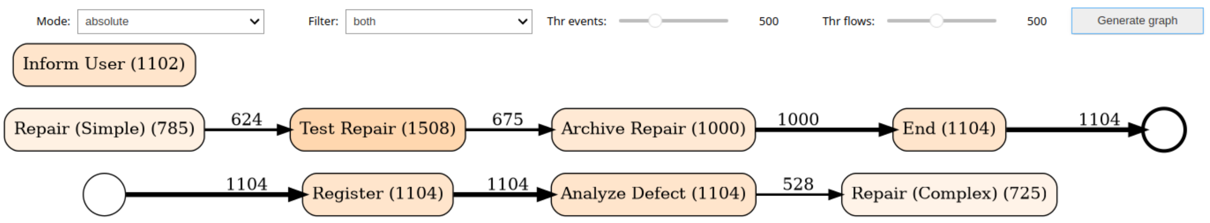

Mode: absolute Filter: both Thr events: 500 Thr flows: 500

Figure 5: Flows $\geq$ 500 and Events $\geq$ 500

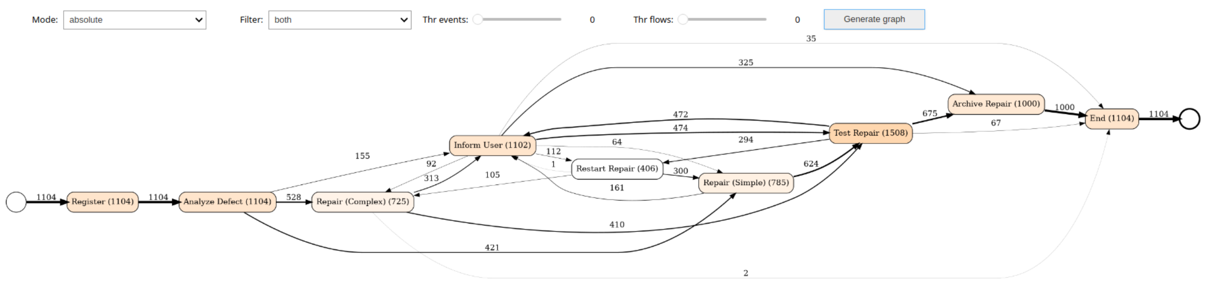

Mode: absolute Filter: both Thr events: 0 Thr flows: 0

Figure 6: Flows $\geq$ 0 and Events $\geq$ 0

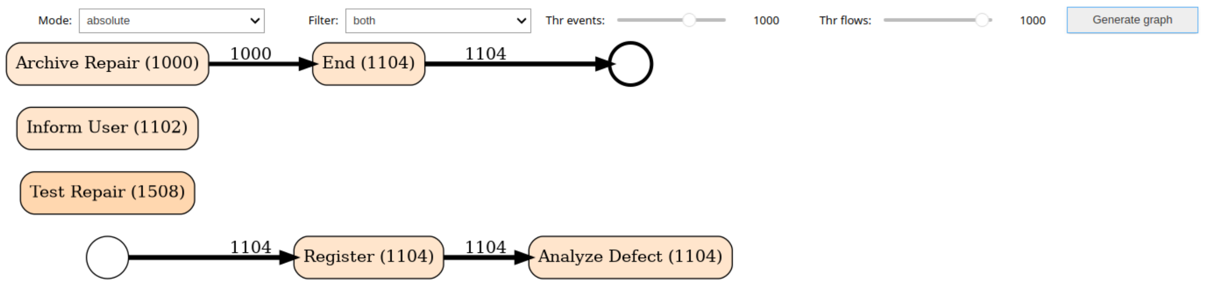

Mode: absolute Filter: both Thr events: 1000 Thr flows: 1000

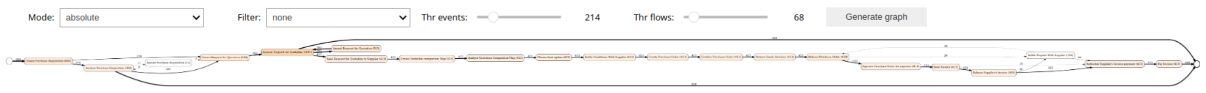
Figure 7: Flows $\geq$ 1000 and Events $\geq$ 1000

4 Task 6

Check how the filtering works for other logs provided in the previous labs. If there is any issue with the model when you filter it with various thresholds, provide the screenshots and explain the problem (at least 3 different problems).

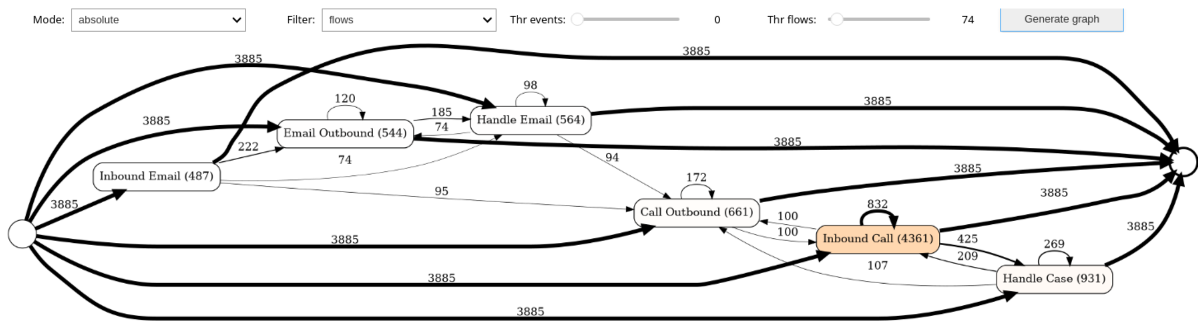
The current implementation if filtering struggles with several issues:

- Naive implementation - the filtering simply removes any events and flows with a count below the threshold, therefore it can create a disconnected graph with lone nodes.
- The images are simple png images, not vector images. Therefore user cannot zoom it or out, unless one would save the image as a separate file and closely inspect it. While it wasn't a problem for the *repairExample.csv*, it's easily noticable in the *purchasingExample.csv* presented in the image 8. To make the image easier to inspect, one has to heavily filter the nodes.
- The automatically generated graph can be confusing to interpret because of the non-intuitive nodes and edges placement. An example of such behavior can be seen in the image 9.
- The program generates a pre-generated image, not a graph. Therefore the user cannot interact with the nodes and change their placement. The nodes are confined to the locations pre-determined by the placement algorithm.



Mode: absolute Filter: none Thr events: 214 Thr flows: 68

Figure 8: Unfiltered *purchasingExample.csv*



Mode: absolute Filter: flows Thr events: 0 Thr flows: 74

Figure 9: Filtered *logExample.csv*

- The *logExample.csv* uses a column header called *Start Date*, as opposed to the *Start Timestamp* of the other two examples.

5 Conclusions

Implementation of our own filtering algorithm proved to be more challenging than expected. To create a proper filtration, one would have to check whether the result graph remains connected. Also a better method of visualization should be used to allow proper analysis and interaction with the graph.

For now we'll probably stick to the Apromore tool.