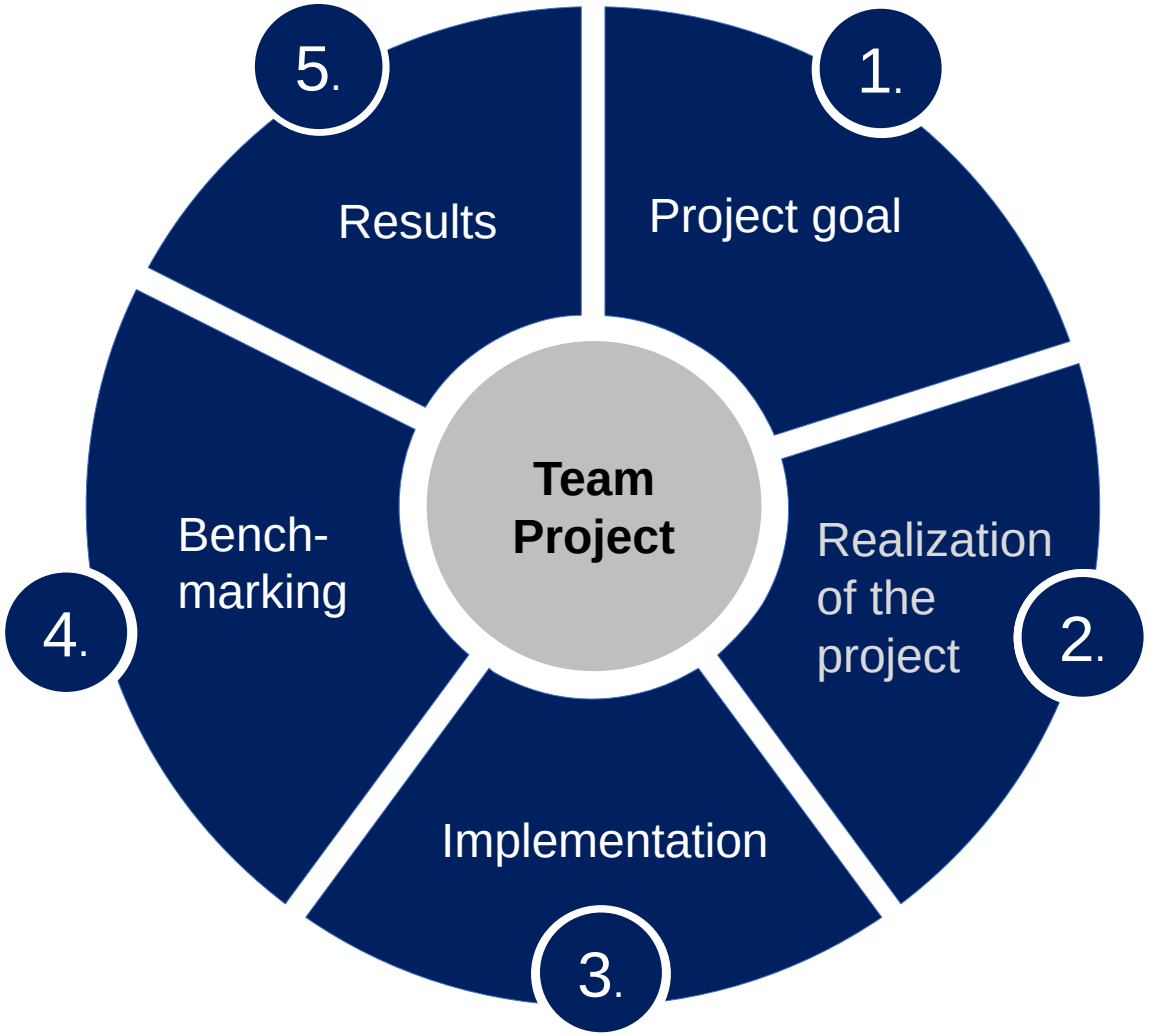
Improvement of the Alpha Algorithm

By Alexander Gerhold, Daniel Häffner, Annalena Sailer,
Alex Sailer, and Marcel-René Wepper

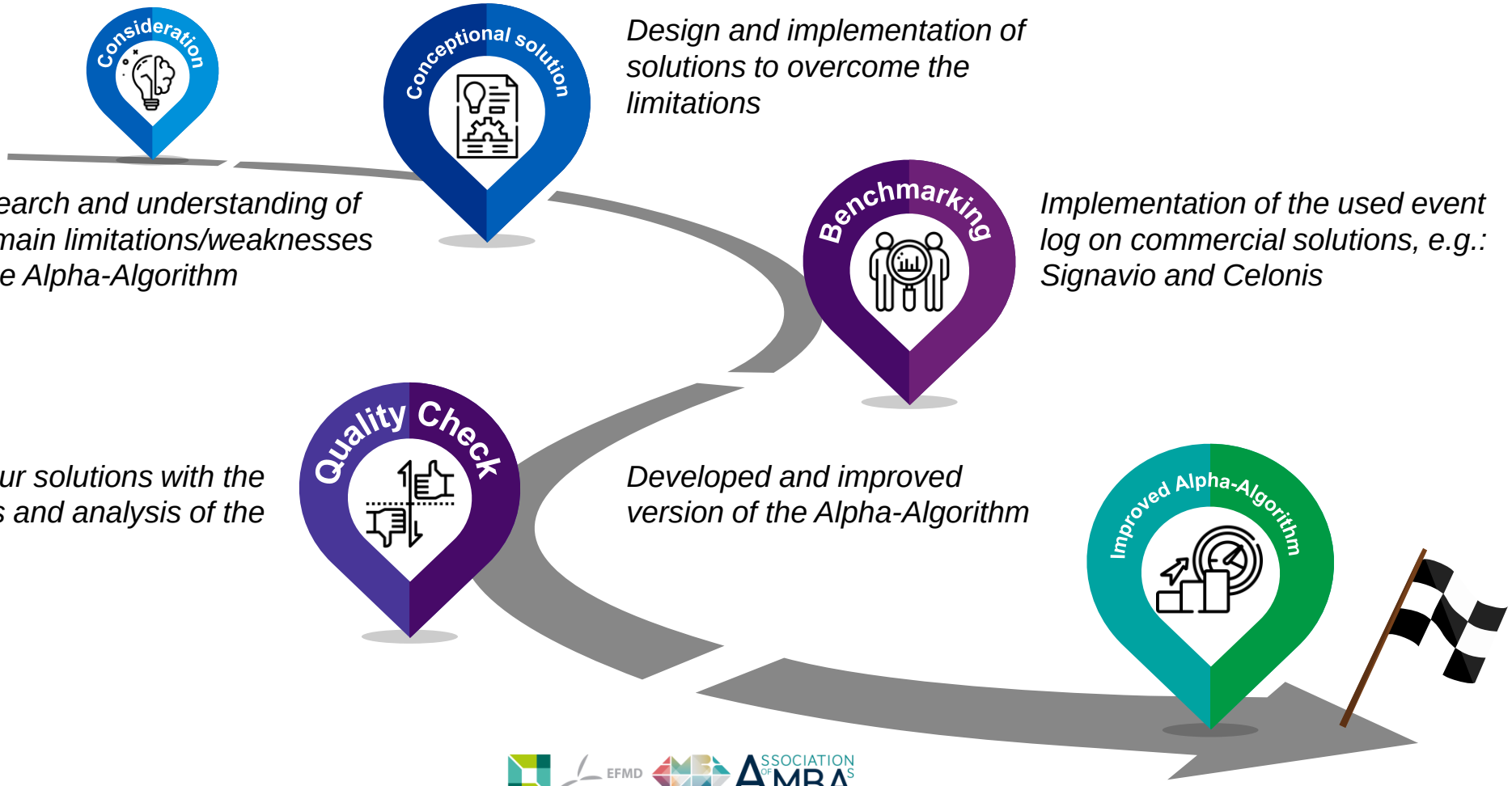
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Agenda


5.
Results
1.
Project goal
Team Project
Realization of the project
2.
4.
Bench-marking
Implementation
3.

1. Overall goal and roadmap of the project

What was the overarching goal of the project?



2. Summary of activities

This underlying chapter is further divided into five different parts

1. Implementation of the Alpha Algorithm	• To begin with, a naive Alpha-Algorithm has been implemented • Several limitations are still in place
2. Research and Selection of Limitations	• In order to come up with viable solutions, several limitations have been researched in order to decide on implementations for limitations
3. Implementation of Improvements	• Concluding the research of the limitations, two limitations were selected in order to solve said limitations
4. Validation of Implementation(s) based on Benchmarking	• In order to validate the improved Alpha-Algorithm, our solution is compared to other, business-oriented process discovery application
5. Finalization phase	• After the implementation and benchmarking, several still underlying limitations are presented, an outlook will be given and a summary presented

2.1 Implementation of the Alpha-Algorithm

Three steps were conducted in order to implement the naive Alpha-Algorithm

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Get Activities from Event Log

L = [<a, b, c, d>,
<a, c, b, d>,
<a, e, d>]

All activities: {a, b, c, d, e}
Start activities: {a}
End activities: {b}

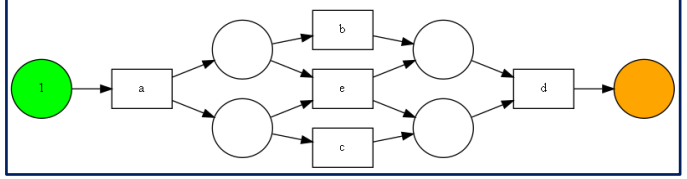
Create Maximal Pairs from Footprint Matrix

	a	b	c	d	e
a	#	→	→	#	→
b	←	#	||	→	#
c	←	||	#	→	#
d	#	←	←	#	←
e	←	#	#	→	#

[({a}, {b, e}), ({a}, {c, e}),
({b, e}, {d}), ({c, e}, {d})]

Draw Petri Net

- Create places for each maximal pair and for start/end activities
- Create an edge from each first item in the pair to its place
- Create an edge from the place to each second item in the pair



2.2 Research and Selection of Limitations

Several limitations were selected in order to improve the Alpha-Algorithm

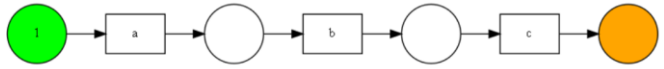
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Loops of length 1

General Problem/Limitation of naive Alpha Algorithm:

The basic Alpha Algorithm is limited in the detection of short loops with length of one.

Lets assume a simple log with only one trace [<a,b,b,c>]. Our implementation of the alpha algorithm generates the following model:



Thus not discovering the length one loop because it can‘t capture the b directly follows b (b->b) relationship and instead assumes unrelatedness (b#b).

Underlying Assumptions for our Solution:

Every length-one loop cannot be connected to the input or output places because it would not be a workflow net otherwise.

Theoretic Solution / Conceptual solution:

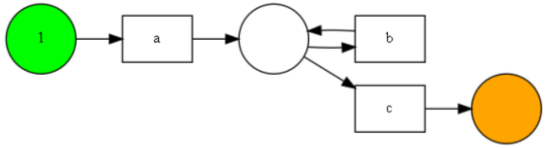
We handle the length-one loop transitionons during pre- and post processeing.

During preprocessing the length-one loops are detected and the position of the transition is stored.

The occurences of length-one loops are now removed and the regular alpha algorithm is perfomed on the remaining log.

Now the transitions and arcs for the length-one loops are reconnected.

For the previous example our implementation now discovers the following model:



2.2 Research and Selection of Limitations

Several limitations were selected in order to improve the Alpha-Algorithm

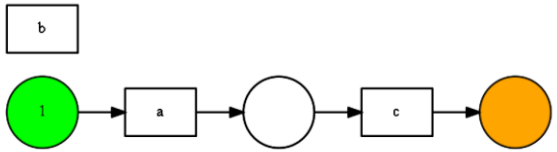
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Loops of length 2

General Problem/Limitation of naive Alpha Algorithm.:

Basic Alpha Algorithm is limited in the detection of short loops with length of two.

Lets assume a simple log with only two traces [<a,b,a,c>, <a,c>]. Our implementation of the alpha algorithm genereates the following model:



Thus not discovering the length-two loop because it can‘t capture the b directly follows a (a->b) relationship and instead assumes unrelatedness (b#b).

Underlying Assumptions:

We assume a loop-complete log, meaning the length-two loops are contained in at least one trace. In the example case this would mean that the sequence *…aba…* occurs at least once.

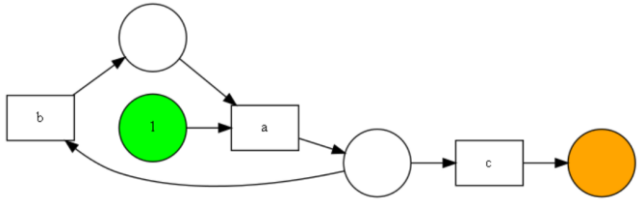
Theoretic Solution / Conceptual solution:

For the handling of length two loops we redefine the definition for the directly follows and parallel relations.

We scan the log for the length two loop pattern (*…aba…).* Activity b now also directly follows activity a if we find this length two pattern.

Also the parallel relation a || b now has the additional condition that the length-two loop pattern (*…aba…)* was not detected in the log.

For the previous example our implementation now discovers the following model:



2.2 Research and Selection of Limitations

Several limitations were selected in order to improve the Alpha-Algorithm

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Noise and Outliers 1/3

General Problem/Limitation of naive Alpha Algorithm:

Noise are infrequent sets of activities which can influence the model output in an unfavorable way. The Alpha Algorithm does not consider any frequencies when extracting the successions from an event log. Thus, a succession $a \rightarrow c$ will be included in the discovered model, even if it appears only once in a large event log.

Underlying Assumptions:

Without further knowledge about the data, it is not possible to distinguish whether noise or outliers are just infrequent or a result of incorrect or incomplete data. This can occur, for example, if the events in an event log are only from a specific time window. The start or the end of a case are being cut, leading to incomplete traces. Anyway, if the time window is large enough, only a small amount of the traces should be affected by this.

For our approach it is assumed, that any infrequent behavior is considered unfavorable and will thus be removed.

Theoretic Solution / Conceptual solution:

We remove infrequent behavior in a pre-processing step. For this, we explored two ways of calculating infrequency:

1. *Global:* Number of traces including a succession relative to the number of traces in an event log.
2. *Per activity:* Number of occurrences of a successor relative to the most frequent successor for an activity (inspired by Inductive Miner Infrequent).

For both infrequency calculations, different approaches to deal with infrequent behavior were implemented:

1. *Removing successor:* This solution can lead to an incomplete petri net, since an activity may no longer be reachable or no longer reach an end activity → such activities have to be removed as well.
2. *Removing full trace:* This solution does not need further changes of the Alpha Algorithm, since only the input gets changed, but can also remove frequent behavior when removing too many traces.

2.2 Research and Selection of Limitations

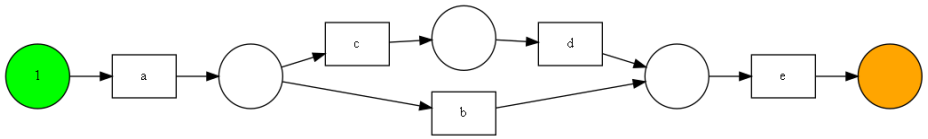
Several limitations were selected in order to improve the Alpha-Algorithm

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

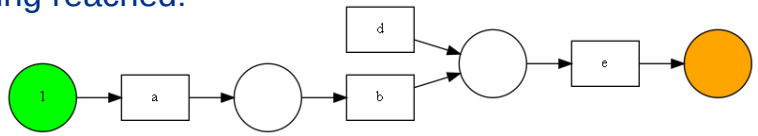
Noise and Outliers 2/3

Exemplary Solution:

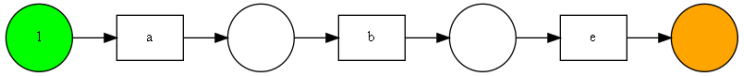
Given the following Petri Net discovered using the naïve Alpha-Algorithm from L = [<a, b, e>², <a, c, d, e>]:



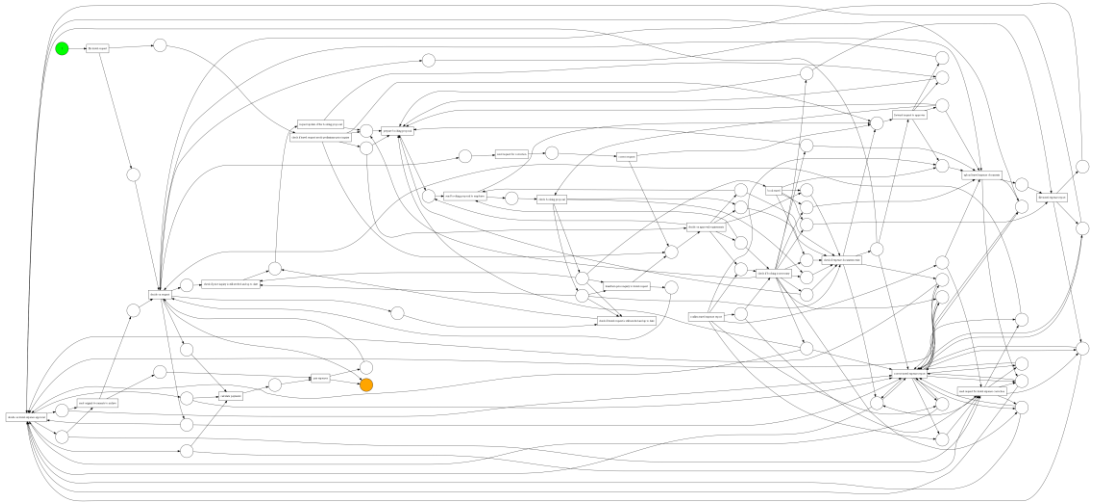
Using the *Per activity* calculation with a threshold of higher than 0.5, *a* → *c* is found to be infrequent. *Removing successor* will lead to *d* not being reached:



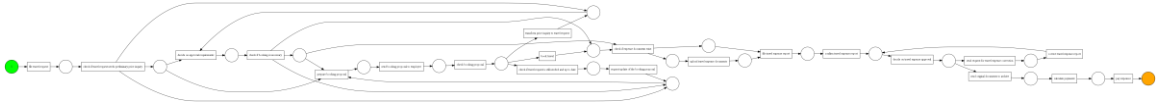
To avoid this, after removing infrequent activities, we also remove activities that are no longer reached from the start place:



For this example, the same could have been achieved by *Removing full trace* or using the *Global* infrequency calculation.



No Filtering applied



Same results for *Global* with 10% and *Per activity* with 20% threshold (for both removal strategies)

(For detailed and more high resolution screenshots, please consider the screenshots in the submission folder.)

2.2 Research and Selection of Limitations

Several limitations were selected in order to improve the Alpha-Algorithm

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Noise and Outliers 3/3

Comparison of approaches:

- Using the Travel dataset (mentioned in section 4. Benchmarking) all our approaches lead to the same *Happy Path* and the same results for most common behavior (1/3 of traces remaining for *Removing full trace*), most likely because the set was already manually cleaned.
- For some cases, *Global* and *Per activity* infrequency calculation lead to different results, e.g. activity *a* has 5 possible successors that are all equally frequent. Using *Global* with threshold higher than 0.2 removes all successions of *a* including *a* since it can no longer reach an end activity. *Per activity* would not remove the successions, since they are all equally frequent.
- For smaller event logs, *Removing successor* is more useful, since *Removing full trace* could remove too much common behavior. For large enough event logs, it can be assumed that the common behavior is captured by the remaining traces.
- *Removing full trace* is faster than *removing successor*, since the following steps are performed on less data and we do not need the step of removing unreachable activities.

Advantages

- Different frequency calculation functions and removing strategies give the user the opportunity to choose what is best for their event log.
- The threshold is variable, giving the user the advantage to decide, how much infrequent behavior should be filtered from the log.

Disadvantages:

- Using *Removing full trace* can quickly lead to empty models, if all traces contain infrequent behavior.
- Selecting a good threshold, to balance simplicity and fitness of the model can be very hard, but is given for any approach of noise filtering.

3. Implementation and Design Choices 1/2

Three steps were conducted in order to solve the task

1. Research

- First of all, everyone of our team conducted its own research in order to get a general idea of the algorithm and how it works
- For this, videos of the lectures by Prof. Dr. Wil van der Aalst as well as Prof. Dr. Jana Rehse have been revisited
- In addition, several papers were used to get the theoretical background of said algorithm and its limitations (see bibliography)

2. Implementation Alpha-Algorithm

- For the implementation, we started with a python notebook Alpha_Algorithm.ipynb, to follow the steps of the alpha algorithm on a set of toy traces and be able to show the intermediate steps (start/end activities, footprint matrix, pairs)
- Then we set up a python project, that provided the implemented alpha algorithm as a starting point to implement further improvements
- Dependencies of the code are defined via conda in the environment.yaml file, gviz needs to be installed seperately for visualization
- We use pandas to load a real-life event log and pm4py to visualize the resulting petri net of our alpha algorithm
- Loading event logs is implemented in the data_loader.py and provides functions for loading traces from excel files as well as one for loading the travel data csv dataset
- The alpha algorithm and its extensions are implemented in alpha_extended_algorithm.py. It can be executed with the *run_alpha_extended_algorithm* function which provides various parameters to enable and control the extensions
- The extended alpha algorithm is based on sub-functions to retreive activity related information and a method to filter traces based on parameters
- The extended alpha algorithm returns an object of the class *AlphaPetriNet* which represents a petri net and allows to show and save the net

3. Implementation and Design Choices 2/2

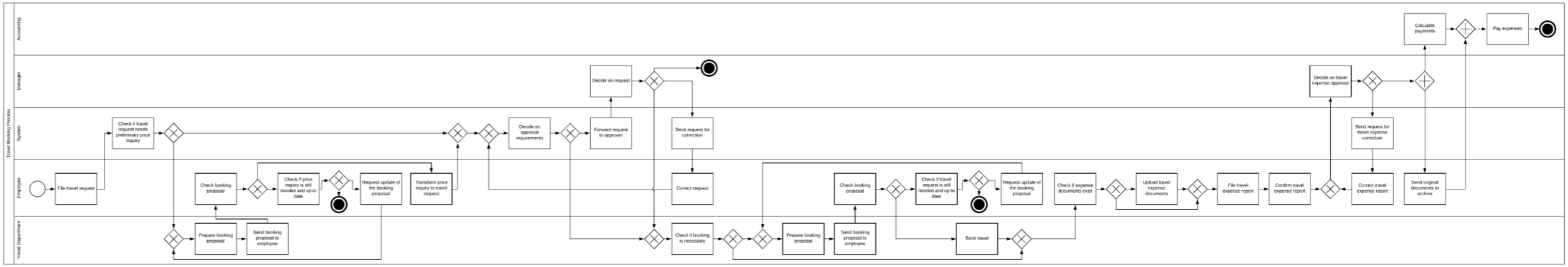
Three steps were conducted in order to solve the task

3. Implementation Limitations

- As suggested in the case description, the naive Alpha-Algorithm contains several limitations, such as loops of length 1 and 2 as well as its weakness for noise and outliers etc.
- In order to come up with reasonable solutions, every limitation proposed by the Use-Case as well as the book has been evaluated and afterwards decided on, which limitations to solve
- The implementations to overcome the limitations are included in alpha_extended_algorithm.py
- They can be activated using parameters in *run_alpha_extended_algorithm*, so each solution can be used on its own or in combination with the other limitation solution
- To run the Alpha-Algorithm with a detection of length 1 and length 2 loops, the *detect_loops* parameter needs to be set to *True*
- For the filtering of noise, several combinations (as mentioned on slide 8) are possible. The filtering is made as a preprocessing step, so in case of *Removing full traces* there are no changes to the Alpha-Algorithm at all, only on the input of the algorithm. For *Removing successor* a placeholder is introduced for the filtered activity in a trace. This placeholder will then be ignored by the Alpha-Algorithm when creating the Footprint Matrix and the Pairs
- *Removing full traces* can be used by setting the *filter_full_traces* parameter to *True*, otherwise *Removing successor* is used
- To use the *Per activity* calculation for frequency, *per_node* parameter needs to be set to *True*, otherwise *Global* calculation is used
- The threshold for noise can be adjusted using *min_support.* When *min_support=None*, no noise filtering will be made
- Examples on how to load data, run the Alpha-Algorithm and use different settings can be found in example.py

4. General Idea, Trace & Selected Competitors

Following, general information of the upcoming chapter can be found



General Idea:

- In order to validate the improved Alpha Algorithm, we are going to compare our result (using *Global, removing full traces* approach) with the results of other commercially available solutions
- This will give a general impression on how well our solution peforms in comparison to other business solutions
- For the comparison, we will compare the Happy Paths of all solutions and one special Use-Case

Trace Information:

- The underlying trace is based on the MobIS 2019 challenge
- Before the trace can be used by our implementation and the selected competitors, it was cleaned up manually (see next slide)
- After the manual clean up for complete cases, the log consists of 55.845 rows, including 3.356 cases with 25 activities in total

Selected Competitors

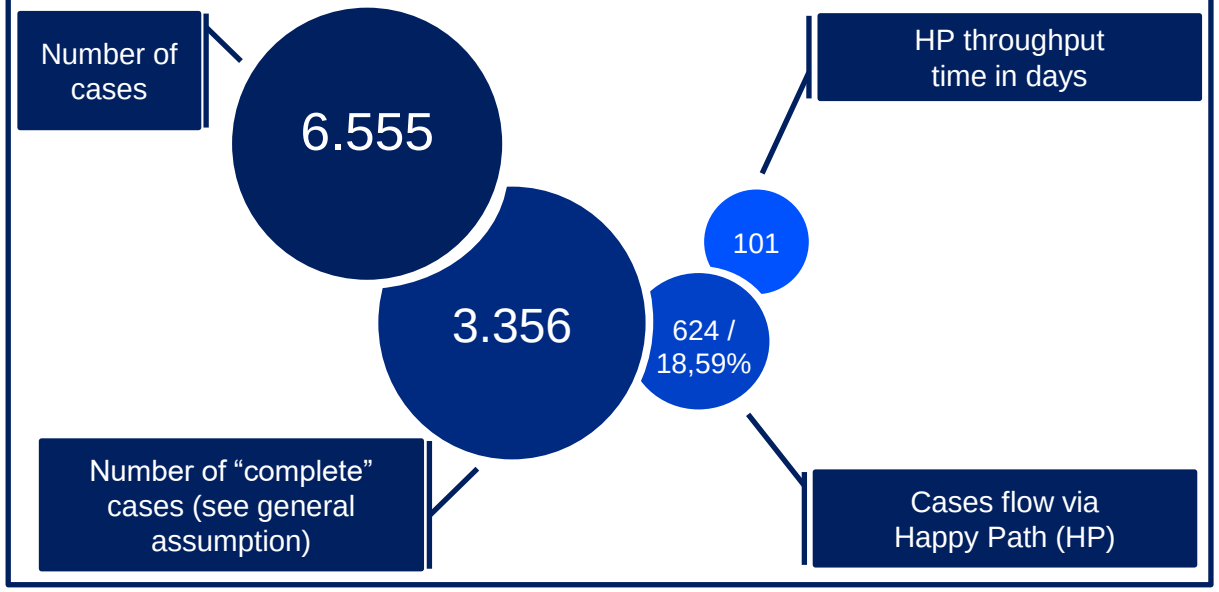
- Following software solutions were chosen:
  - Celonis, as being one of the biggest companies with having Process Mining as its core strength
  - Signavio, as it was mentioned in the course and its recent acquisition by SAP

4.1 Trace Data for Validation

On the following slide information of the log can be found

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

General overview of data set



General assumptions

- Considered 3.356 cases; starting at "file travel request" and ending after "pay expenses" are considered complete cases and used for further investigation

Tools used

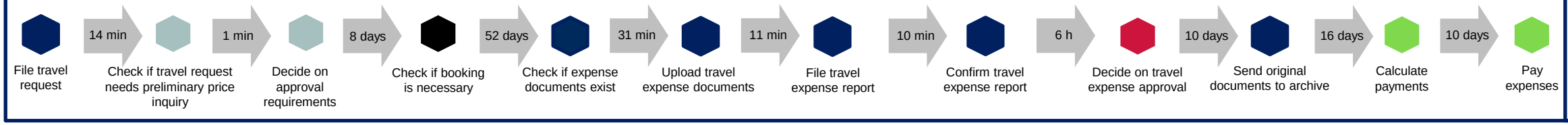
Excel for manual pre-processing

Python + Jupyter for implementation

Celonis as benchmark

Signavio as benchmark

Happy Path

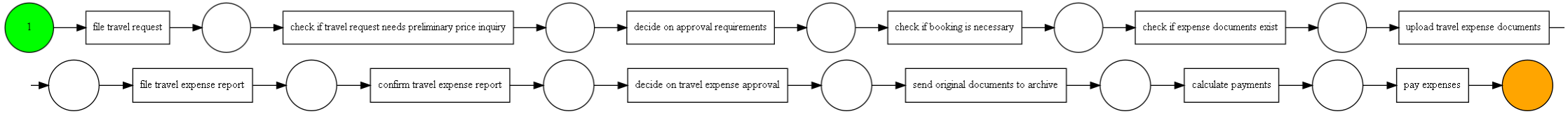


4.2 Results – Happy Path

As a basic comparison, the proposed Happy Path was used...

1. Implemented Alpha Algorithm

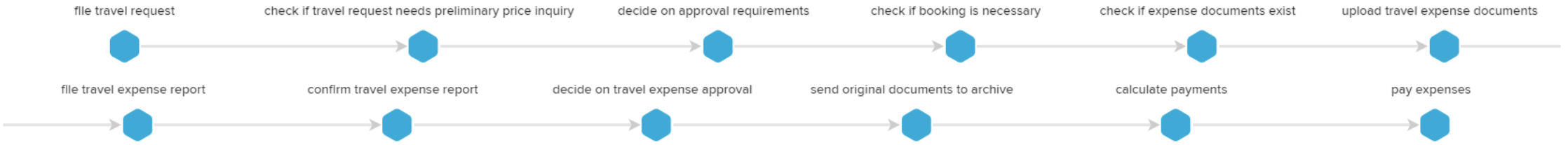
- With a threshold of 45% (using *Global, Removing full trace*), we are able to display the Happy Path proposed on the previous slide
- In total, 624 cases are still present – leading to 18.59% of all considered cases following the Happy Path, with only 12 activities left in the Happy Path



2. Celonis

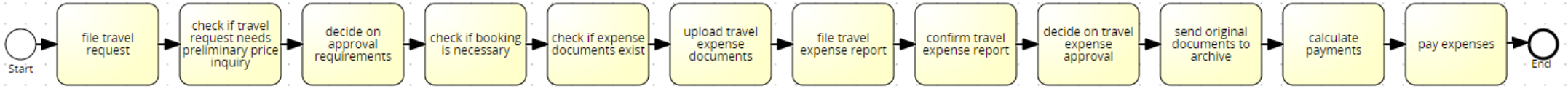
- With 73% of the activities and 63% of connections, the default Happy Path is displayed
- Celonis computed Happy Path is contained in 16.66%, leading to 559 cases in total, with a total of only 12 activities remaining

Algorithmic happy path



3. Signavio

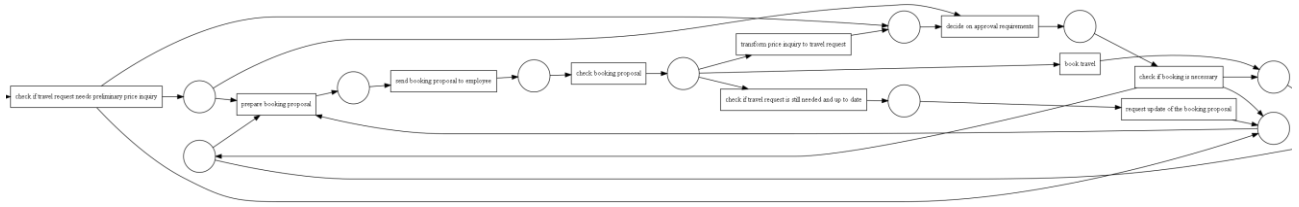
- Signavio was able to compute the same Happy Path, which is contained in 13.94%, leading to 468 cases in total



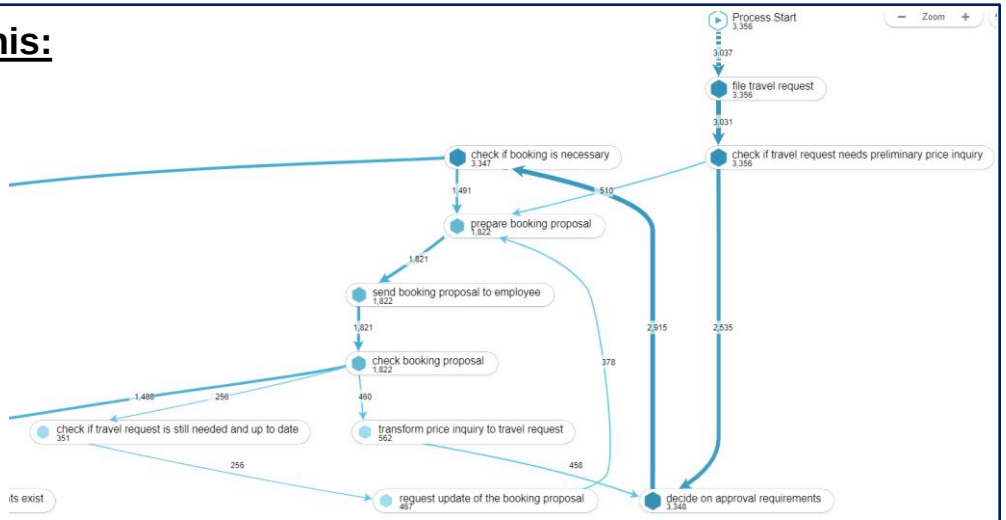
4.3 Exemplary detailed comparison

After a rather superficial comparison, one specific Use-Case is compared

Implementation:



Celonis:



Description of Implementation and Celonis:

When looking at this specific part of the model, after “check if travel request needs...”, multiple paths can be chosen: Either directly to “decide on approval”, to prepare booking proposal or via the loop caused by “check booking proposal”.

Comparison:

When comparing both solutions, the implementation with a threshold of 10% and Celonis solution with 98.2% of activities and 92.4% of connections, identical results in regards to the process can be found. When comparing other aspects such as the loop on “decide on travel expense approval”, very similar solutions can be found (not displayed in this figure).

Preliminary Conclusion:

When choosing similar threshold levels for our implementation and the amount of connections in Celonis, looking at this exact position of the discovered model, the same results in regards to the process as well as the logic can be found. This is also due to the fact, that we already cleaned the data manually, since the given event log was from a specific time window and included incomplete cases. Despite the fact, that only a few and selected use-cases were used in order to compare our solution to other business solutions (due to space limitation), it is very clear that our proposed solutions is very well capable of creating a valuable solution in the region of process discovery. (For detailed and more high resolution screenshots, please consider the screenshots in the submission folder.)

5.1 Remaining Limitations and possible improvements

Limitations of the Alpha-Algorithm which were out of scope for our project

Soundness

- Resulting model does not need to be a sound WF-net
- Assumption in Alpha-Algorithm
- There exist some algorithms which ensure the soundness of a model

Non-local dependencies

- Resulting from non-free choice process constructs
- Foundational limitation
- Alpha-Algorithm can not handle difficult constructs like mixture of choice and synchronization

Incompleteness

- It can not be assumed to have seen all possibilities in the event log (unrealistic)
- Foundational limitation
- Completeness refers to the problem of having too little data
- Alpha-Algorithm assumes weak notion of completeness

Representional bias

- No discovery of transitions with duplicate or invisible labels
- Possible solutions is including a silent transition, but it is not within the representational bias of the Alpha-Algorithm

Implicit places

- Can be easily removed without affecting the firing sequences
- Do not add value
- Log behaviour can be replayed without these places
- Implicit places only complicate matters

5.2 Summary – And future Work

What we did and what could be improved …

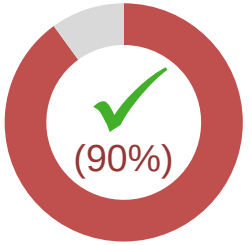
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Implementation of Alpha Algorithm

- Traces can be correctly displayed within in the scope of the Alpha Algorithm
- Highlights the limitations

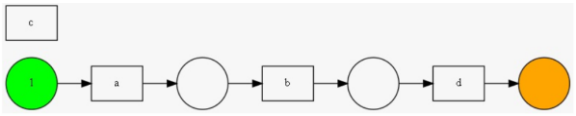
Handling of noise activities and traces

- Different approaches regarding frequency calculation and removal of infrequent behavior
- Future work:
  - Further checks for edge cases among different approaches
  - Improvement of runtime



Handling of length 1 and 2 loops

- Standard and the most edge cases can be correctly displayed by our model
- Exception: Some special edge cases like loop of length 1 inside loop of length 2 <a,b,c,b,b,c,b,d>



- Future work:
  - Improvement of runtime
  - Handling of edge cases

Benchmarking: Comparison with State-of-the-Art Process Mining tools

- Signavio
- Celonis

Statement regarding independence

We hereby guarantee that the present paper is my own and that I have properly acknowledged any help or support received from other individuals. Furthermore, we declare that neither this work nor parts of it have been submitted by myself or others to another university as part of a formal degree requirement. All intellectual property of others is clearly cited as such.
All secondary literature and other sources are certified and listed in the bibliography. The same applies for graphic illustrations, pictures and all internet sources.
We agree that my work may be electronically saved and sent anonymized to be checked for plagiarism. I am aware that this paper cannot be graded if this declaration is not signed.

Team 6
Mannheim, 02. July 2021

Bibliography

Aalst, Wil van der. *Process Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. https://doi.org/10.1007/978-3-662-49851-4.

Cheng, Hsin-Jung, and Akhil Kumar. “Process Mining on Noisy Logs — Can Log Sanitization Help to Improve Performance?” *Decision Support Systems* 79 (November 2015): 138–49. https://doi.org/10.1016/j.dss.2015.08.003.

Weijters, A. “Process Mining: Extending the Alpha-Algorithm to Mine Short Loops,” 2004.

Submission

Link to GitHub: https://github.com/sailera19/TeamProjectAPM