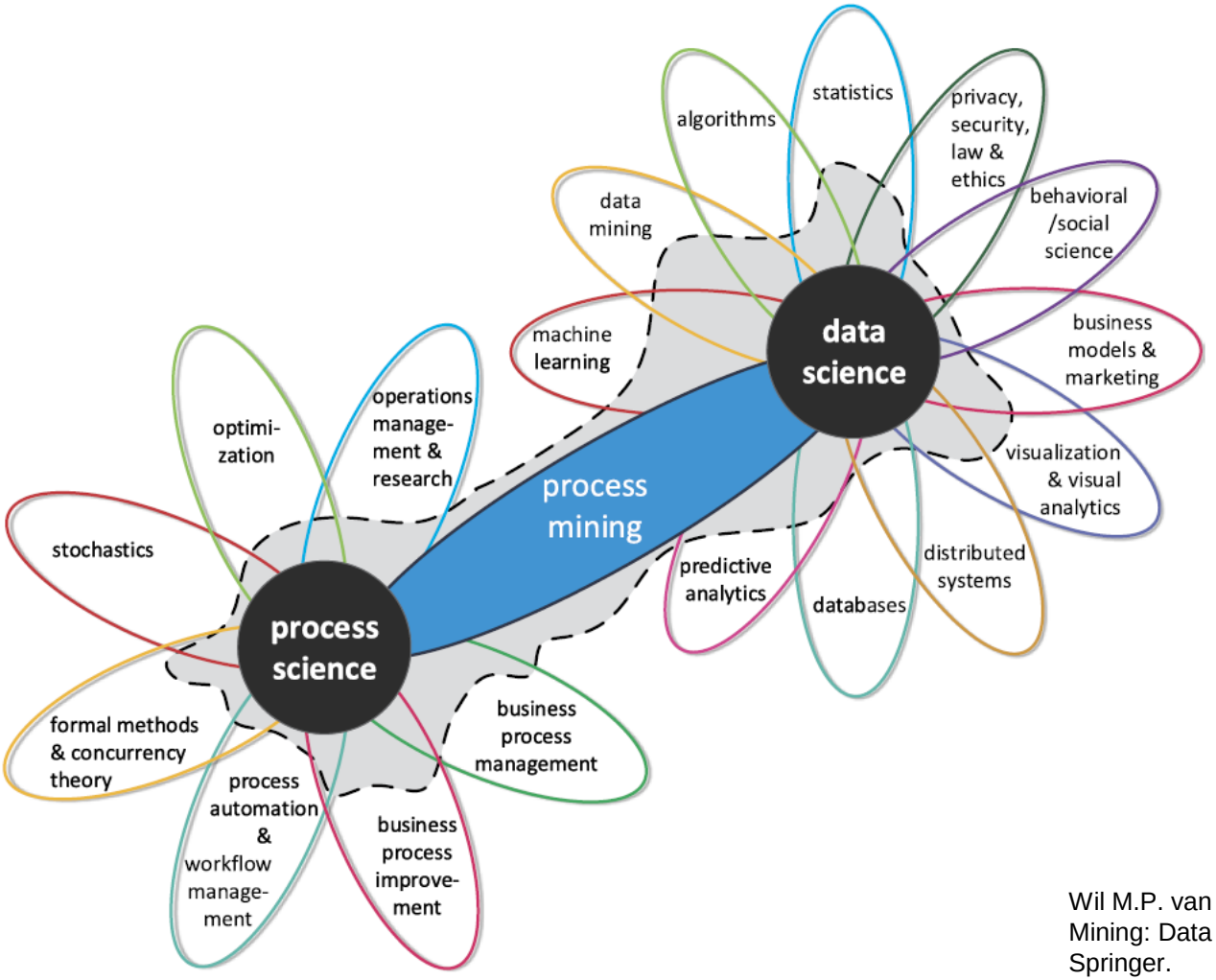
Efficient Construction of Behavior Graphs for Uncertain Event Data

Marco Pegoraro, Merih Seran Uysal, Wil M.P. van der Aalst

RWTH Aachen University, Germany

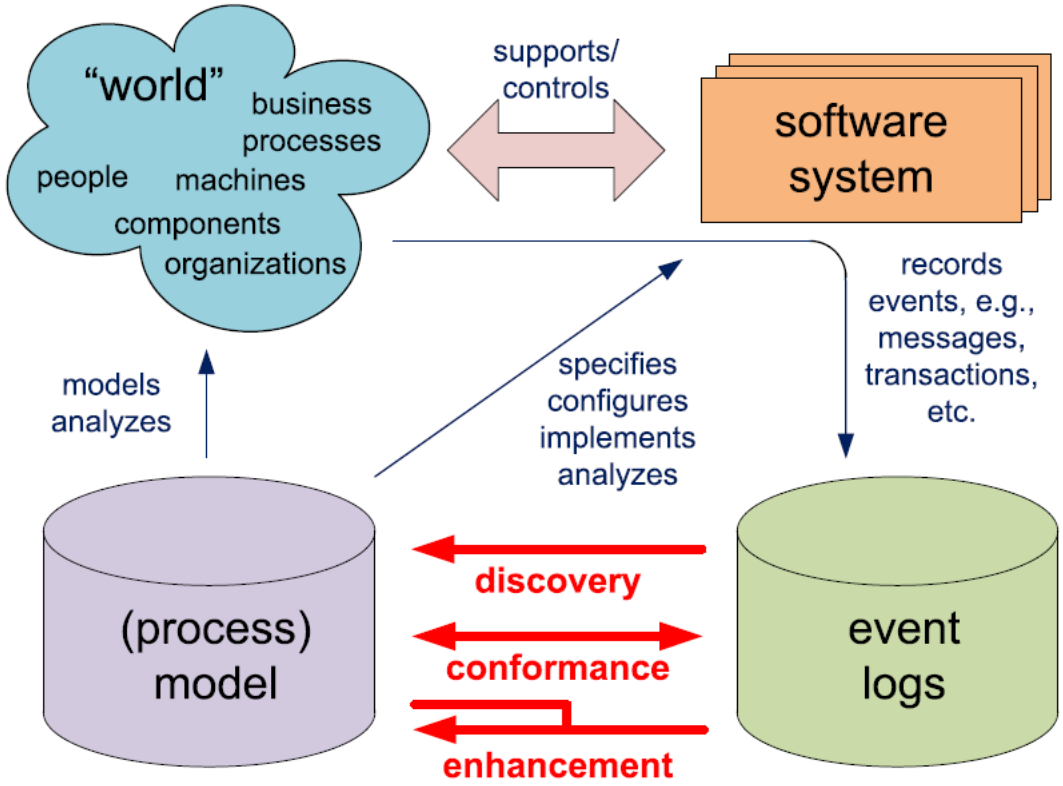
23rd International Conference on Business Information Systems
June 9th 2020

Process Mining

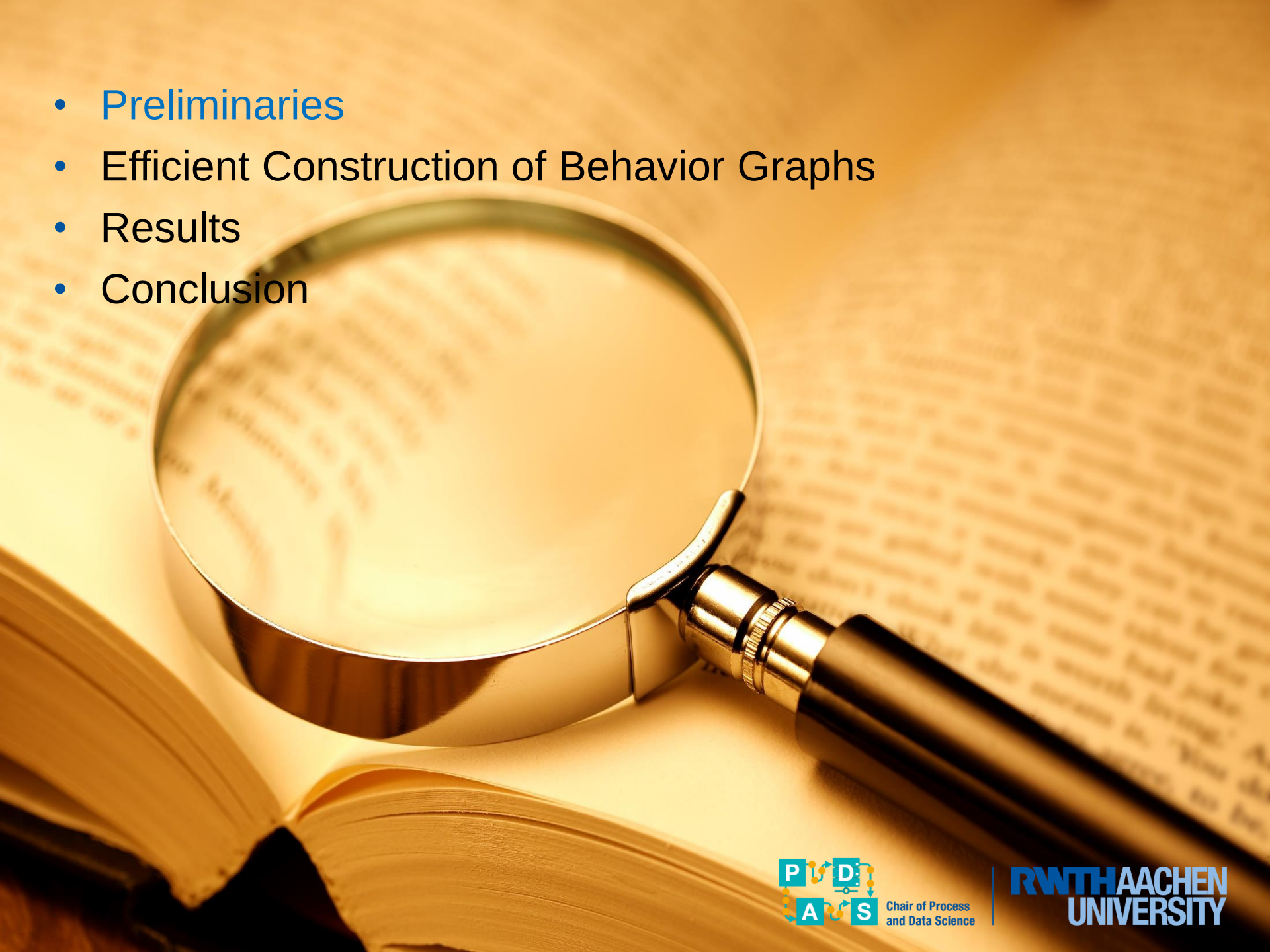


Wil M.P. van der Aalst. “Process Mining: Data science in action." Springer.

Process Mining



Wil M.P. van der Aalst. "Process Mining: Data science in action." Springer.



- Preliminaries
- Efficient Construction of Behavior Graphs
- Results
- Conclusion

Event log

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	register request	Pete	50	...
	35654522	30-12-2010:15.06	examine casually	Mike	400	...
	35654524	30-12-2010:16.34	check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	decide	Sara	200	...
	35654526	06-01-2011:12.18	reinitiate request	Sara	200	...
	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	...
	35654530	08-01-2011:11.43	check ticket	Pete	100	...
	35654531	09-01-2011:09.55	decide	Sara	200	...
	35654533	15-01-2011:10.45	pay compensation	Ellen	200	...

Wil M.P. van der Aalst. “Process Mining: Data science in action." Springer.

Traces

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...

A **trace** consists in the set of all the events belonging to a certain case.
It is usually represented by a sequence ordered by timestamp:

1. <register request, examine thoroughly, check ticket, decide, reject request>
2. <register request, check ticket, examine casually, decide, pay compensation>

Example of uncertain trace

An **uncertain trace** is a process trace where some attributes are described with a **range or a set of possible values**.

In this paper, we consider traces with uncertainty on the **timestamp attribute**.

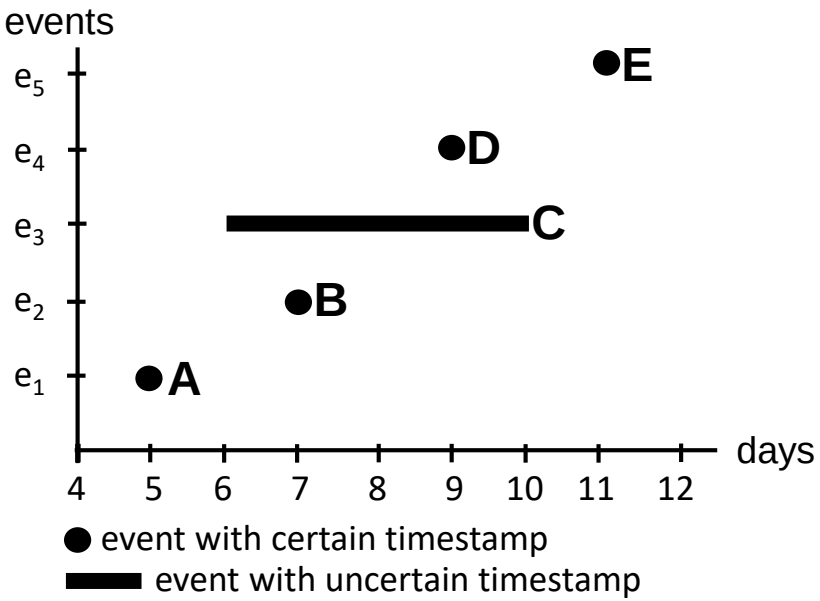
Event ID	Case ID	Timestamp	Activity
e_1	0	05.12.2011	A
e_2	0	07.12.2011	B
e_3	0	**[06.12.2011, 10.12.2011]**	C
e_4	0	09.12.2011	D
e_5	0	11.12.2011	E

The exact timestamp of e_3 belongs to this interval

Marco Pegoraro and Wil M.P. van der Aalst, "Mining Uncertain Event Data in Process Mining," *2019 International Conference on Process Mining (ICPM)*, Aachen, Germany, 2019, pp. 89-96. doi: 10.1109/ICPM.2019.00023.

Realizations of an uncertain trace

Event ID	Case ID	Timestamp	Activity
e_1	0	05.12.2011	**A**
e_2	0	07.12.2011	**B**
e_3	0	[06.12.2011, 10.12.2011]	**C**
e_4	0	09.12.2011	**D**
e_5	0	11.12.2011	**E**



Possible realizations:
<A, B, C, D, E>
<A, B, D, C, E>
<A, C, B, D, E>

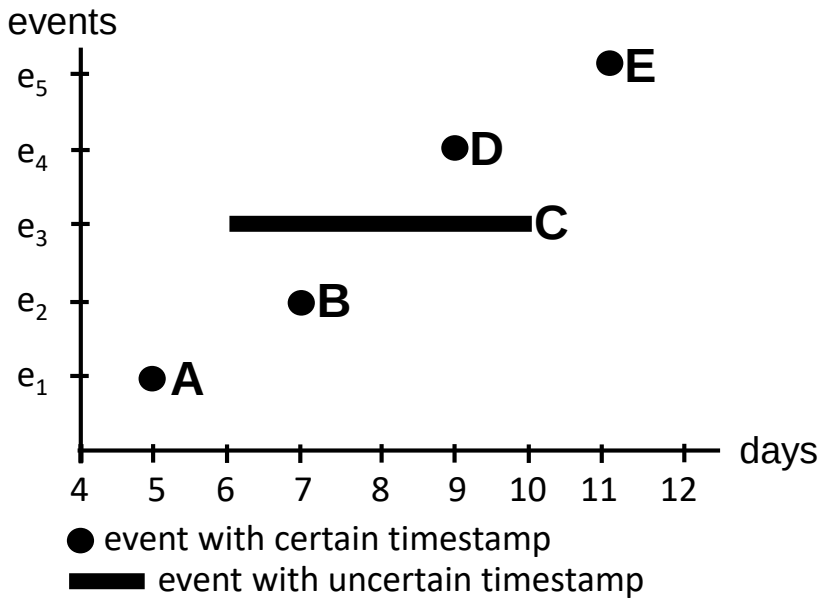
Representation of an uncertain trace

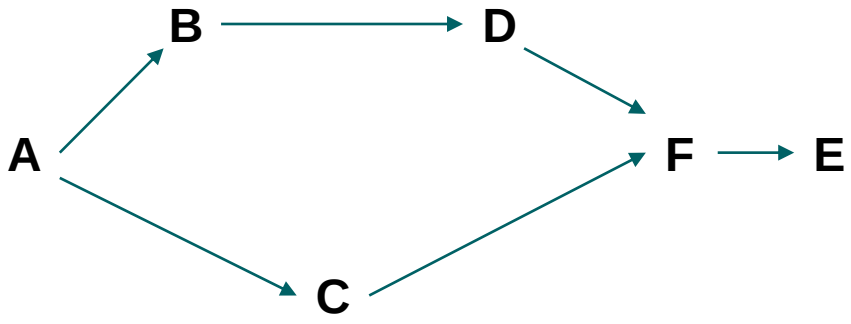
Uncertain traces cannot be represented by a single sequence like regular ones.

Instead, they need to be represented by a **graph** that shows **precedence relationships**.

Behavior graph

Event ID	Case ID	Timestamp	Activity
e_1	0	05.12.2011	**A**
e_2	0	07.12.2011	**B**
e_3	0	[06.12.2011, 10.12.2011]	**C**
e_4	0	09.12.2011	**D**
e_5	0	11.12.2011	**E**





Behavior graph

Obtaining the behavior graph is essential to performing process mining on uncertain event logs.

Behavior graphs allow to query for relationships between activities in the process, a fundamental step of **process discovery** [1].

They are also important to be able to compare the possible behavior of an uncertain trace and a reference model, enabling **conformance checking** over uncertainty [2].

[1] Marco Pegoraro, Merih Seran Uysal, and Wil MP van der Aalst. "Discovering Process Models from Uncertain Event Data." *International Conference on Business Process Management*. Springer, Cham, 2019.
[2] Marco Pegoraro and Wil M.P. van der Aalst, "Mining Uncertain Event Data in Process Mining," *2019 International Conference on Process Mining (ICPM)*, Aachen, Germany, 2019, pp. 89-96. doi: 10.1109/ICPM.2019.00023.

Behavior graph creation

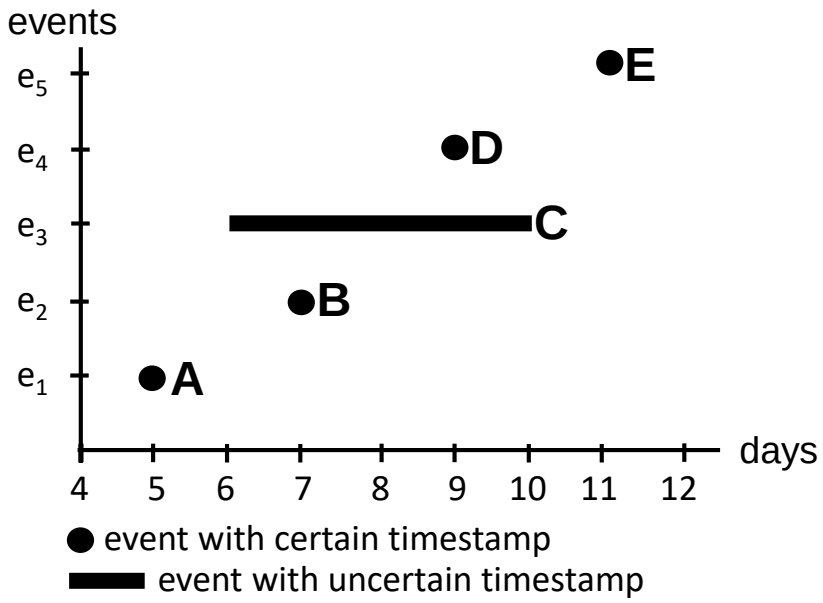
A behavior graph is conceptually simple to obtain:

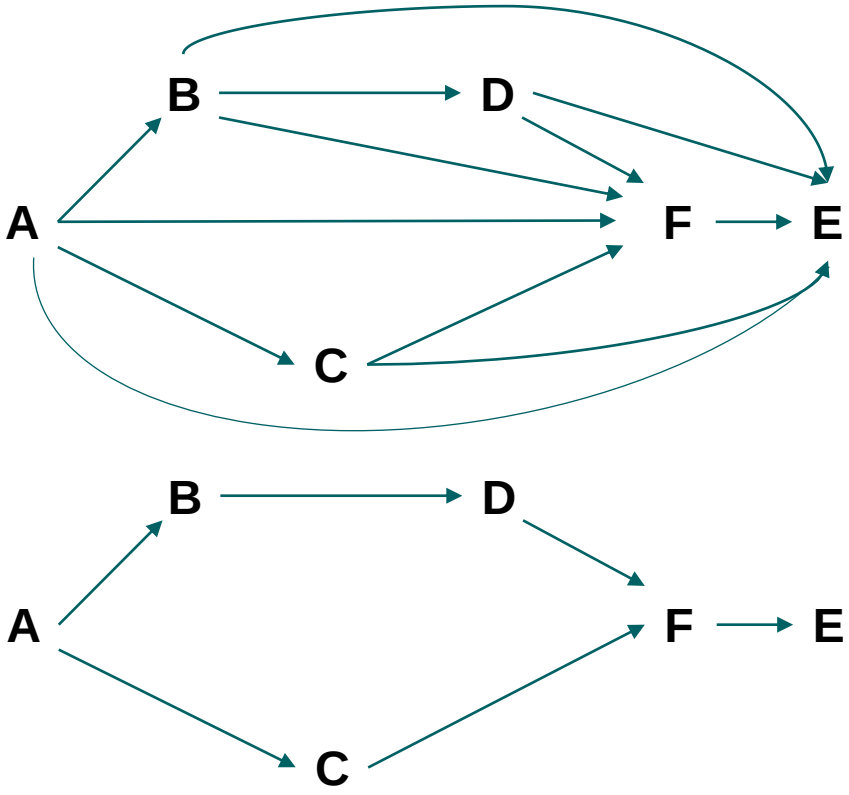
- Connect event e_1 to event e_2 if they do not overlap and e_1 happened before e_2
- Perform transitive reduction on the resulting graph

Transitive reduction: removing the maximum number of edges from a graph without altering the **reachability** between nodes.

Behavior graph creation

- Connect event e_1 to event e_2 if they do not overlap and e_1 happened before e_2
- Perform transitive reduction on the resulting graph





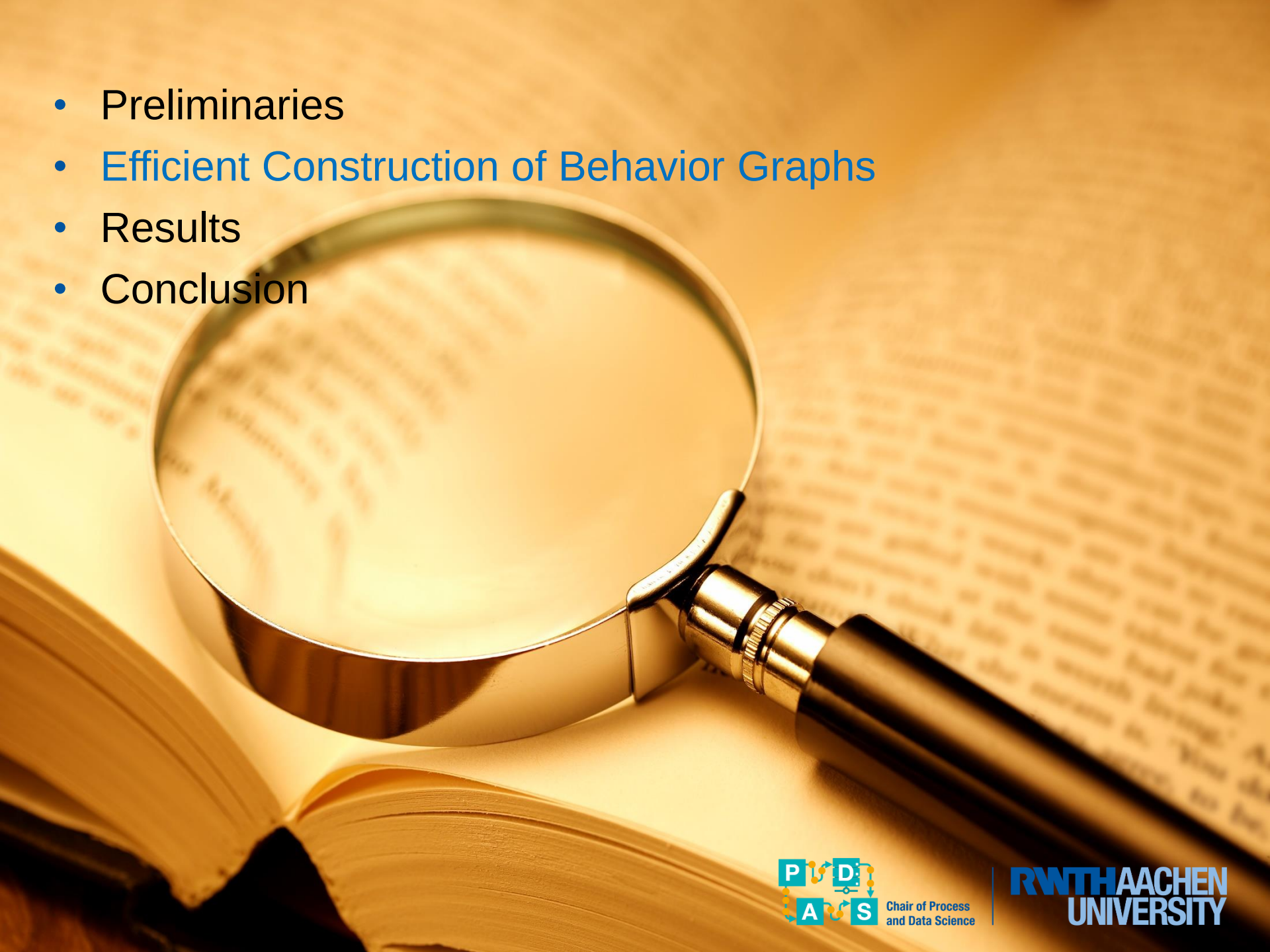
Behavior graph creation

- Connect event e_1 to event e_2 if they do not overlap and e_1 happened before e_2
- Perform transitive reduction on the resulting graph

Transitive reduction: removing the maximum number of edges from a graph without altering the **reachability** between nodes.

Cost for a trace with n events: **$O(n^3)$**

- Preliminaries
- Efficient Construction of Behavior Graphs
- Results
- Conclusion



Preprocessing

- Preprocessing step: transform the uncertain trace in a list of activities sorted by timestamp

- In this list, events with an uncertain timestamps become **two entries:** the left and right extreme of the time interval are inserted in the list, labeled as such, with a reference to the original activity label

EventID	Case ID	Timestamp	Activity
e1	0	05.12.2011	A
e2	0	07.12.2011	B
e3	0	[06.12.2011, 10.12.2011]	C
e4	0	[08.12.2011, 11.12.2011]	D
e5	0	09.12.2011	E
e6	0	[12.12.2011, 13.12.2011]	F

Preprocessing

- Preprocessing step: transform the uncertain trace in a list of activities sorted by timestamp

- In this list, events with an uncertain timestamps become **two entries:** the left and right extreme of the time interval are inserted in the list, labeled as such, with a reference to the original activity label

EventID	Case ID	Timestamp	Activity
e1	0	5	A
e2	0	7	B
e3	0	[6, 10]	C
e4	0	[8, 11]	D
e5	0	9	E
e6	0	[12, 13]	F

Preprocessing

- Preprocessing step: transform the uncertain trace in a list of activities sorted by timestamp

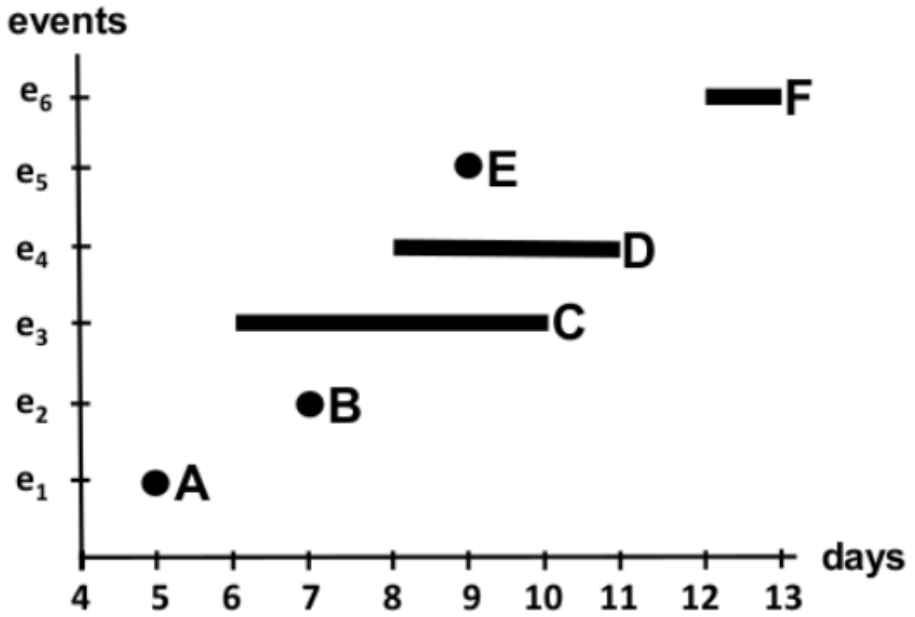
- In this list, events with an uncertain timestamps become **two entries:** the left and right extreme of the time interval are inserted in the list, labeled as such, with a reference to the original activity label

EventID	Case ID	Timestamp	Activity
e1	0	5	A
e2	0	7	B
e3	0	[6, 10]	C
e4	0	[8, 11]	D
e5	0	9	E
e6	0	[12, 13]	F

A, C_L, B, D_L, E, C_R, D_R, F_L, F_R

Preprocessing

Visiting the list in order is the equivalent of **sweeping** the time diagram in order to «discover» beginning and end of events.



A, C_L, B, D_L, E, C_R, D_R, F_L, F_R

Behavior graph creation algorithm

Input:

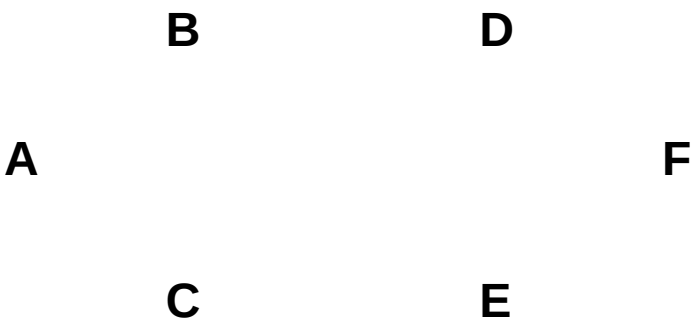
list *The preprocessed list of timestamps*

```
for i = 1 to n
   if list[i] is not a left extreme
   for j = i+1 to n
      if list[j] is a left extreme
         connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
         connect list[i] node with list[j] node and stop
      if list[j] is a right extreme
         if list[i] node was already connected with list[j] node
            stop
         else
            continue
```

Behavior graph creation: example

$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

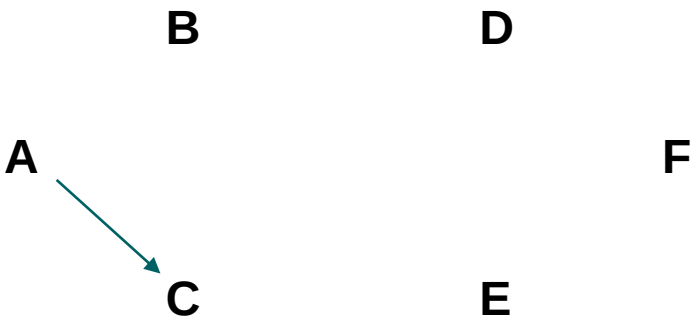
```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```



Behavior graph creation: example

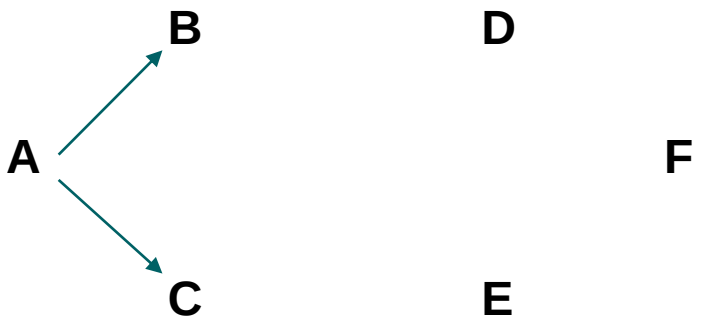
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```



Behavior graph creation: example

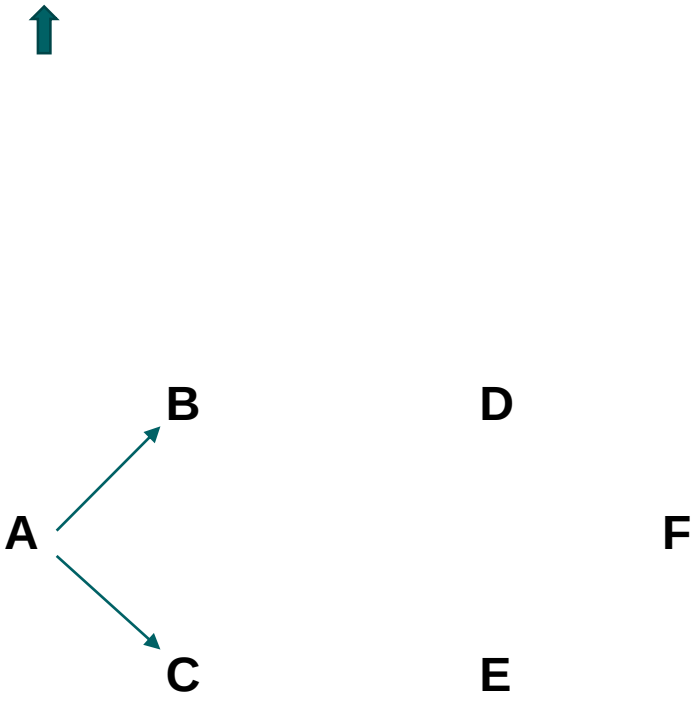
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

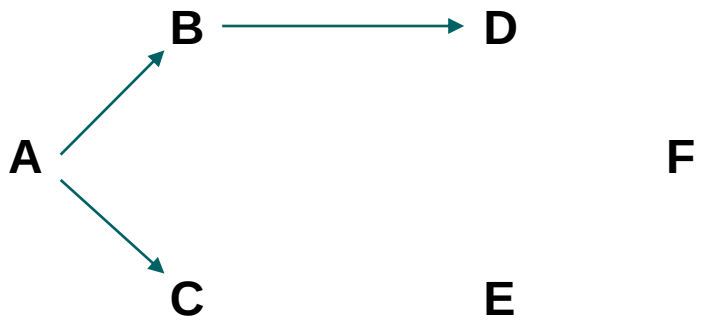
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

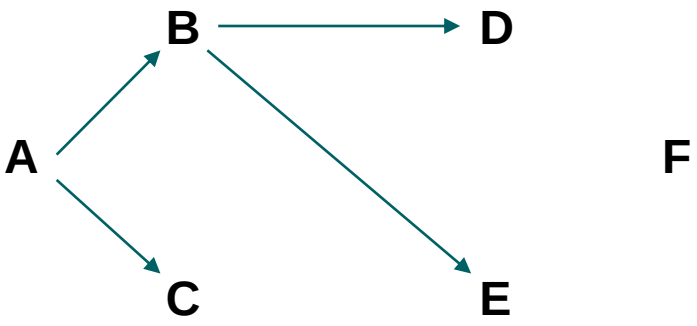



```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

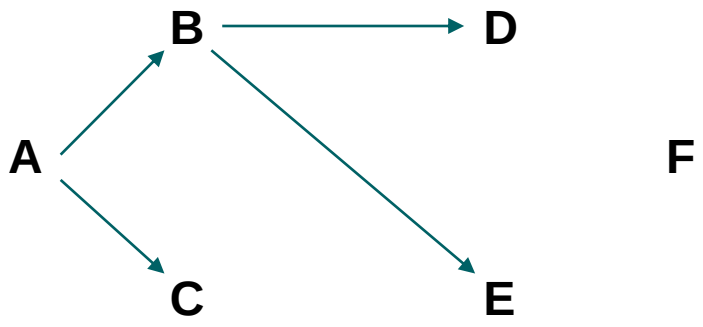
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```



Behavior graph creation: example

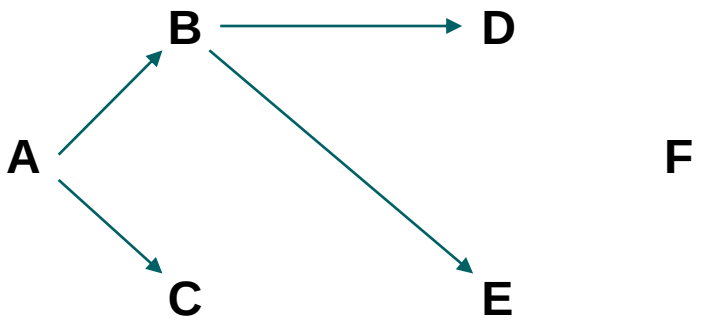
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

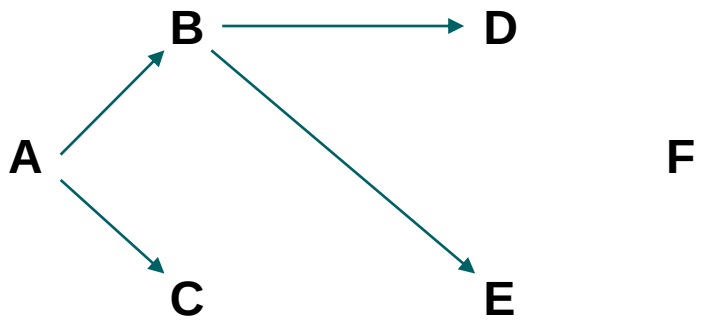
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

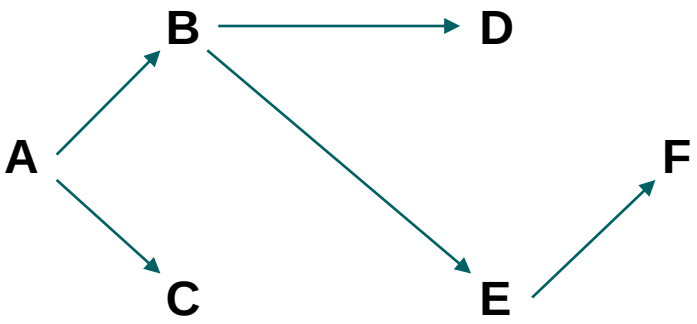



```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

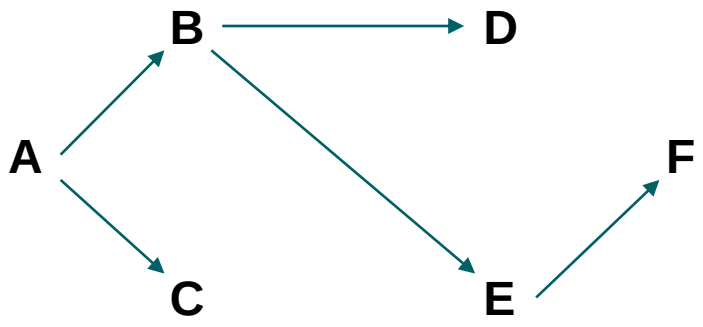
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```



Behavior graph creation: example

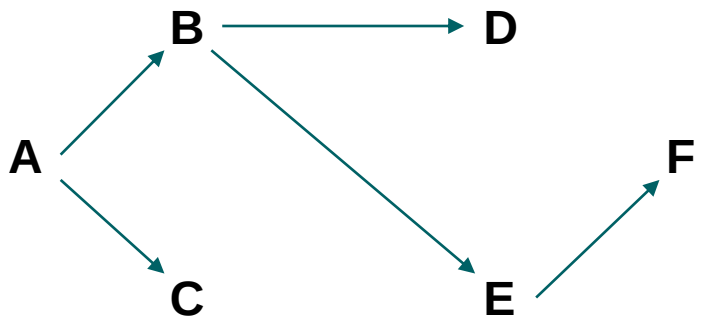
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

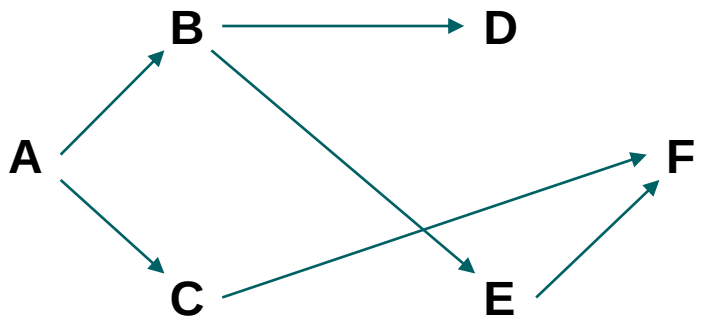
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

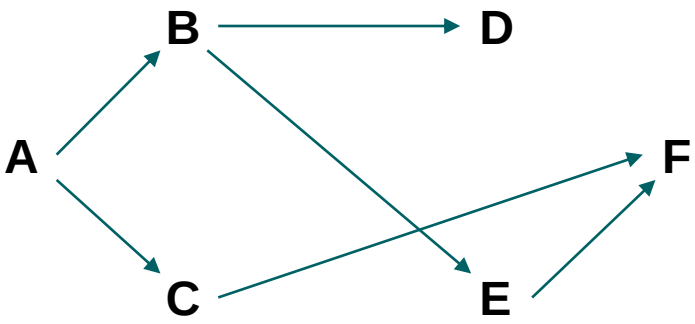



```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

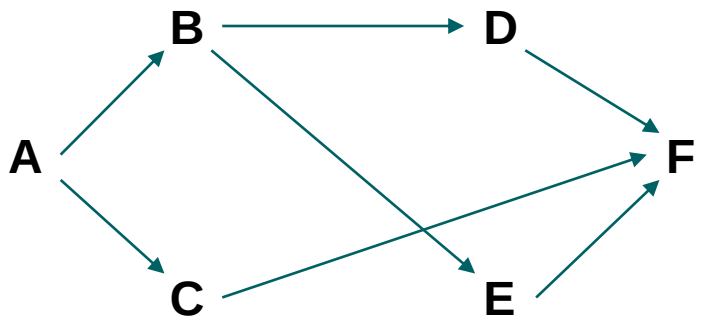
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$

```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```



Behavior graph creation: example

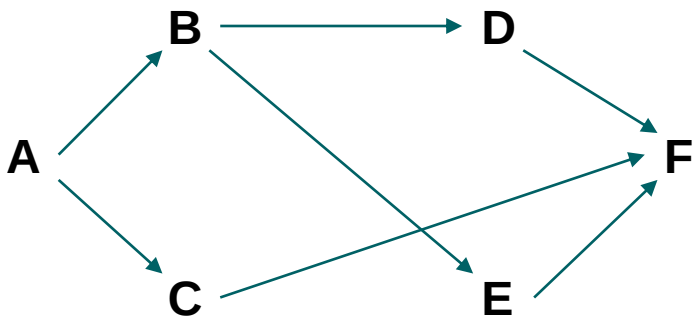
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

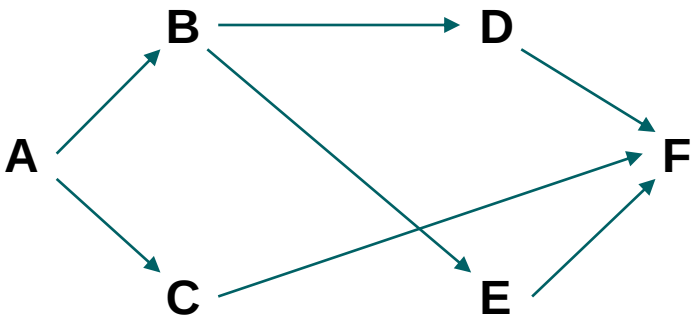
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

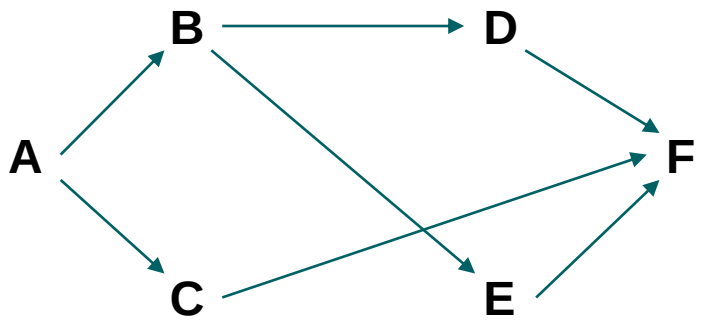
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

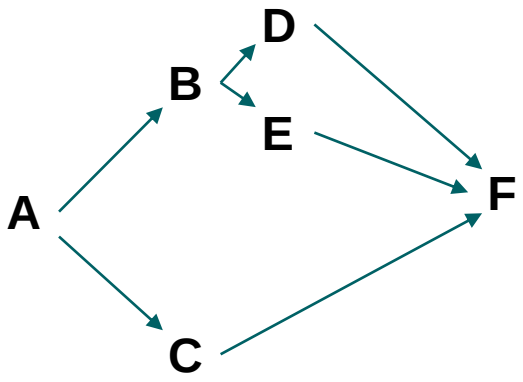
$A, C_L, B, D_L, E, C_R, D_R, F_L, F_R$




```
for i = 1 to n
    if list[i] is not a left extreme
    for j = i+1 to n
        if list[j] is a left extreme
            connect list[i] node with list[j] node and continue
        if list[j] is a certain timestamp
            connect list[i] node with list[j] node and stop
        if list[j] is a right extreme
            if list[i] node was already connected with list[j] node
                stop
            else
                continue
```

Behavior graph creation: example

EventID	Case ID	Timestamp	Activity
e1	0	5	A
e2	0	7	B
e3	0	[6, 10]	C
e4	0	[8, 11]	D
e5	0	9	E
e6	0	[12, 13]	F



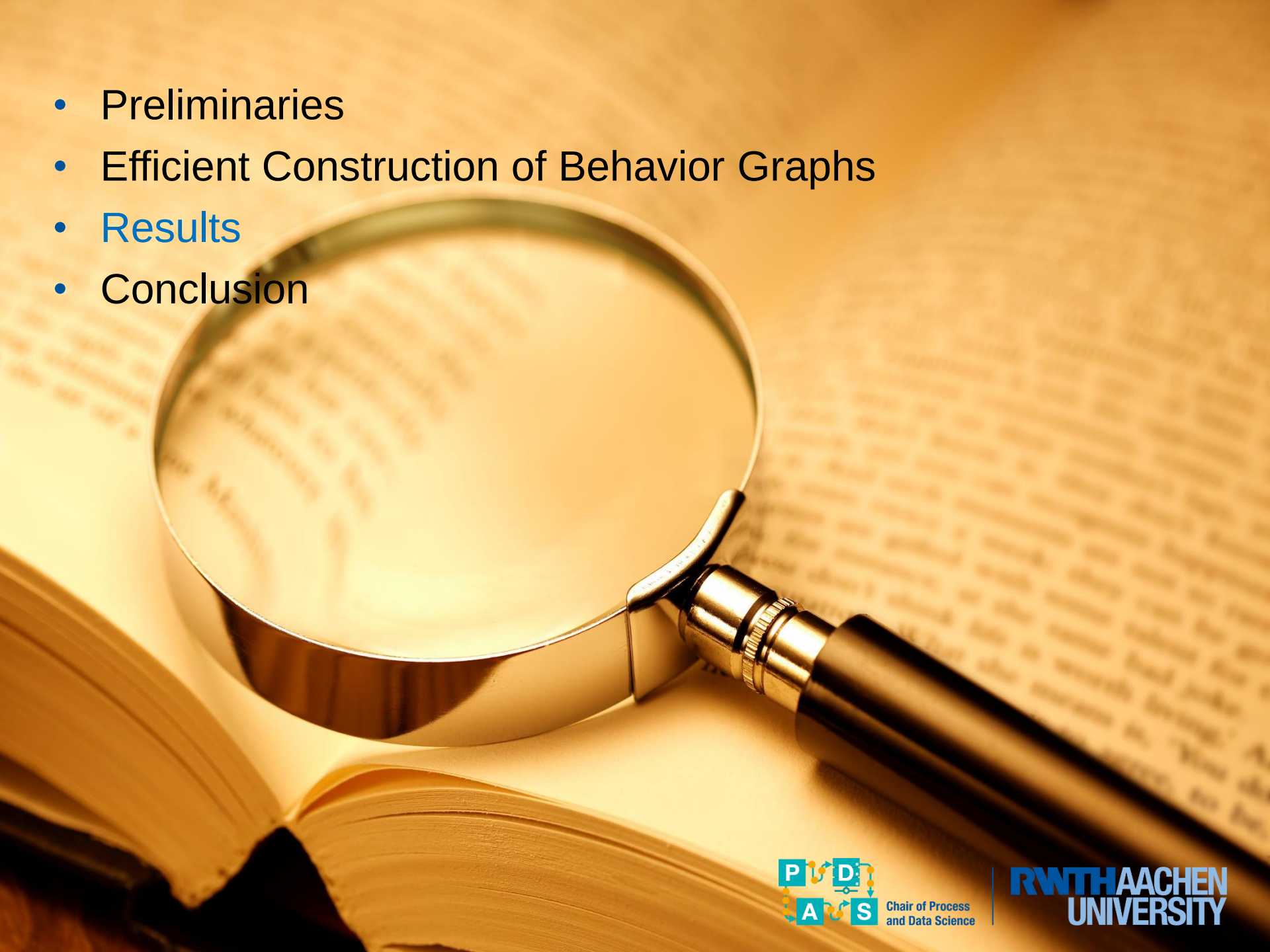
Behavior graph creation: complexity

Let us examine the complexity of this algorithm.

If the uncertain trace has *n* events, the sorted list of timestamp is long at most *2n*.

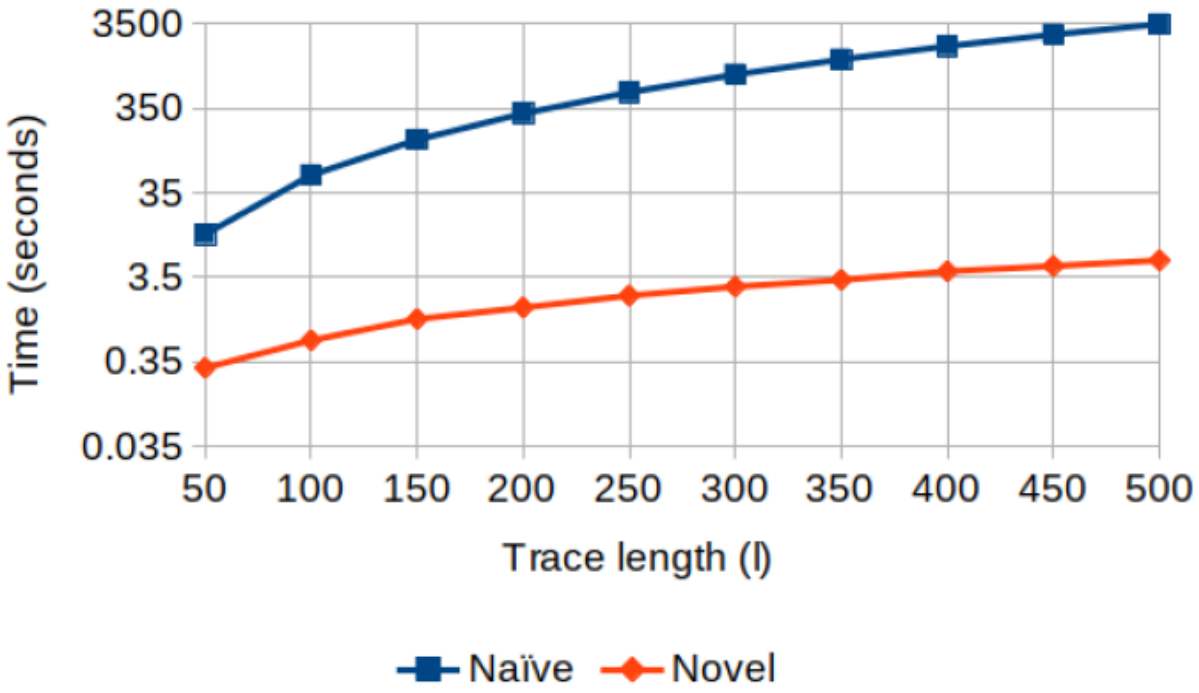
The algorithm sorts this list and then uses two nested loops on it.

The complexity is then $O(2n \cdot \log(2n) + 2n \cdot 2n) =$ **$O(n^2)$.**



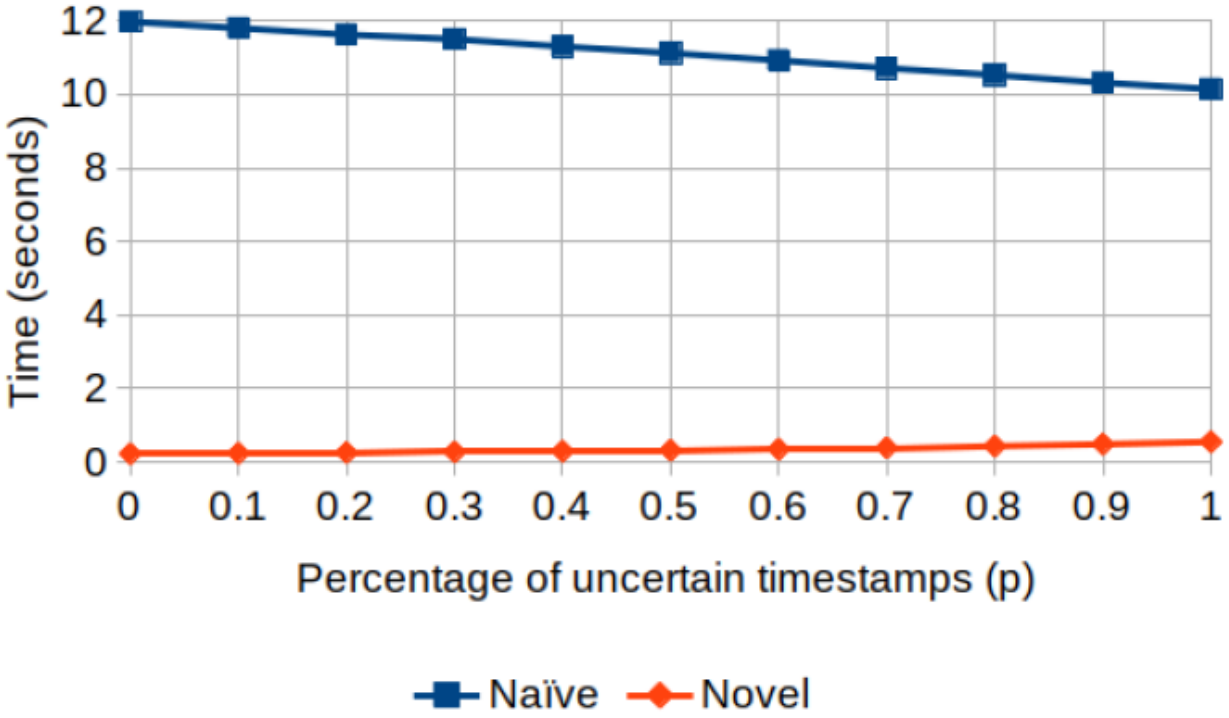
- Preliminaries
- Efficient Construction of Behavior Graphs
- Results
- Conclusion

Behavior graph creation: experiments

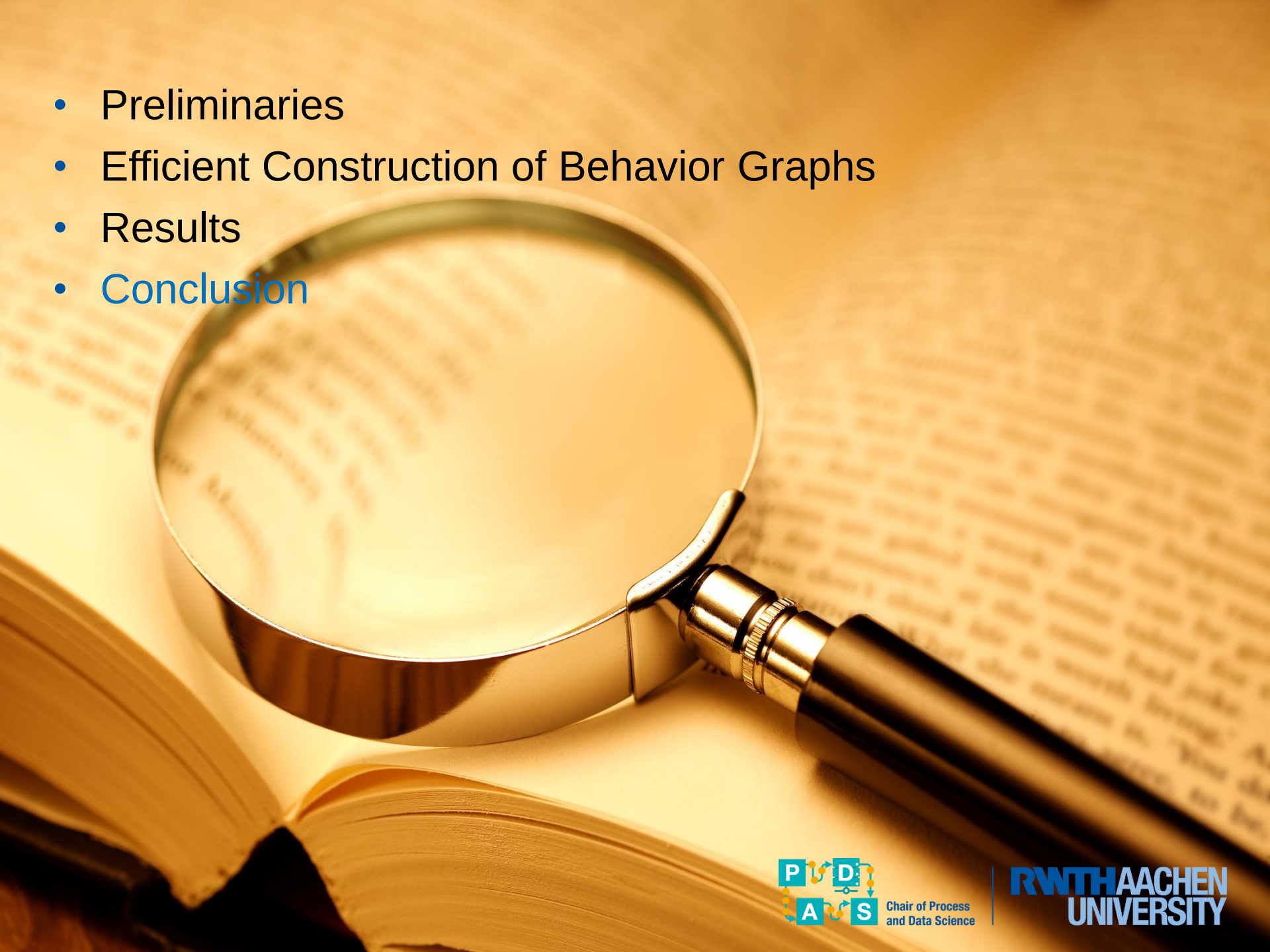


Execution time on an **uncertain log**, in function of the **trace length** (number of traces and number of uncertain events remain constant)

Behavior graph creation: experiments



Execution time on an **uncertain log**, in function of the **percentage of uncertain events** (number of traces and trace length remain constant)



- Preliminaries
- Efficient Construction of Behavior Graphs
- Results
- Conclusion

Conclusion

- The new method of construction for behavior graphs removes a **major bottleneck** in the performance of uncertainty analysis in process mining.

- The running time for conformance and discovery under uncertainty is **strongly reduced** (for typical log dimensions).

- More work needed:
  - Average case analysis
  - More experiments (also in context of discovery and conformance checking)

Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Powered by PM4PY pm4py.org

Experiments available at
https://github.com/proved-py/proved-core/tree/Efficient_Construction_of_Behavior_Graphs_for_Uncertain_Event_Data

Marco Pegoraro
pegoraro@pads.rwth-aachen.de
www.mpegoraro.net

Thank you!

References

Marco Pegoraro and Wil M.P. van der Aalst, "Mining Uncertain Event Data in Process Mining," *2019 International Conference on Process Mining (ICPM)*, Aachen, Germany, 2019, pp. 89-96. doi: 10.1109/ICPM.2019.00023.