



Uncertainty in Process Mining

Marco Pegoraro

My story



- Preliminaries and Definitions
- Conformance Checking over Uncertainty
- Behavior Graph Construction
- Conclusion

Uncertainty in Event Logs

An **uncertain event log** is an event log where some of the values include some sort of quantified uncertainty.

We consider just the *control-flow perspective*: timestamp (continuous value), case id, and activity (discrete values).

Uncertainty can be defined not only on the *attribute* level, but also on the *event* level.

Uncertainty in Event Logs

There can be many high-level sources of uncertainty in event data:

- **Incorrectness:** errors happened while recording data or manipulating the logs (e.g. while merging logs)
- **Coarseness:** variability of an attribute caused by imprecision of a measure (e.g. limitation of sensors)
- **Ambiguity:** the event data is recorded in a way that needs interpretation (e.g. event data recorded as free text)

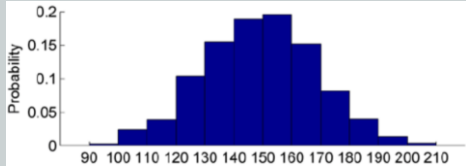
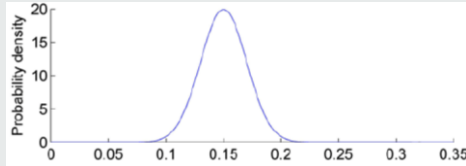
Uncertainty in Event Logs

Very often, we have **coarseness on the timestamp attribute**.

Mainly because of two reasons:

- Data formats **too coarse** (e.g., timestamps recorded with the date but not the time)
- **Recording of events in batches** (e.g., a doctor that inputs data in an information system at the end of the round of visits)

Uncertainty - Taxonomy

	Weak uncertainty	Strong uncertainty
Discrete data	<p>Discrete probability distribution</p> 	<p>Set of possible values</p> $\{x, y, z, \dots\}$
Continuous data	<p>Probability density function</p> 	<p>Interval</p> $\{x \in \mathbb{R} a \leq x \leq b\}$

Uncertainty - Taxonomy

Uncertainty on the *attribute* level:

- **Case ID:** discrete
- **Activity:** discrete
- **Timestamp:** continuous

Uncertainty - Taxonomy

Uncertainty on the *event* level:

An **indeterminate event** is an event that has been recorded as happening only with a certain probability.

Namely, there is the possibility that the event has been recorded but did not happen at all.

Can be seen as a **relaxed version of uncertainty on case ID**.

Example of strongly uncertain trace

Case ID	Timestamp	Activity	Indet. event
{0, 1}	2011-12-05T00:00	A	!
0	2011-12-07T00:00	{B, C, D}	!
0	[2011-12-06T00:00, 2011-12-10T00:00]	D	?
0	2011-12-09T00:00	{A, C}	!
{0, 1, 2}	2011-12-11T00:00	E	?

Example of weakly uncertain trace

Case ID	Timestamp	Activity	Indet. event
{0:0.9, 1:0.1}	2011-12-05T00:00	A	!
0	2011-12-07T00:00	{B:0.7, C:0.3}	!
0	$\mathcal{N}(2011-12-08T00:00, 2)$	D	?:0.5
0	2011-12-09T00:00	{A:0.2, C:0.8}	!
{0:0.4, 1:0.6}	2011-12-11T00:00	E	?:0.7

«Uncertainty is not* a mistake»

- Embed it in techniques
- Avoid direct filtering/repairing
- Avoid resorting to Maximum Likelihood Estimations
- Probabilities: Bayesian interpretation (rather than frequentist)

*necessarily

Nature of Uncertainty

The concept of uncertainty that we apply is connected to **subjective probability** (e.g. Bayesian) rather than objective (frequentist).

Event log 1:

$[<a, b, d>^{70}, <a, c, d>^{30}]$

Event log 2:

$[<a, \{b:0.7, c:0.3\}, d>^{100}]$

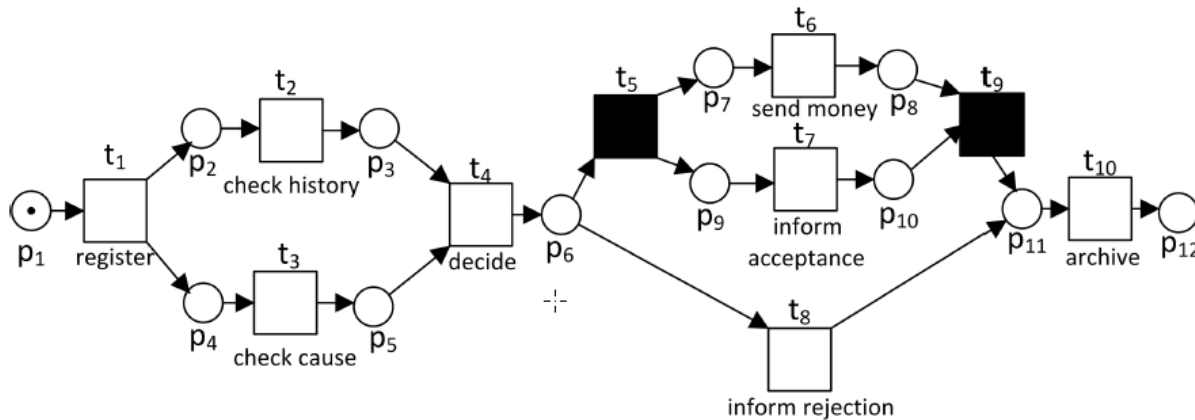
- Preliminaries and Definitions
- Conformance Checking over Uncertainty
- Behavior Graph Construction
- Conclusion

Conformance Checking in Uncertain Settings

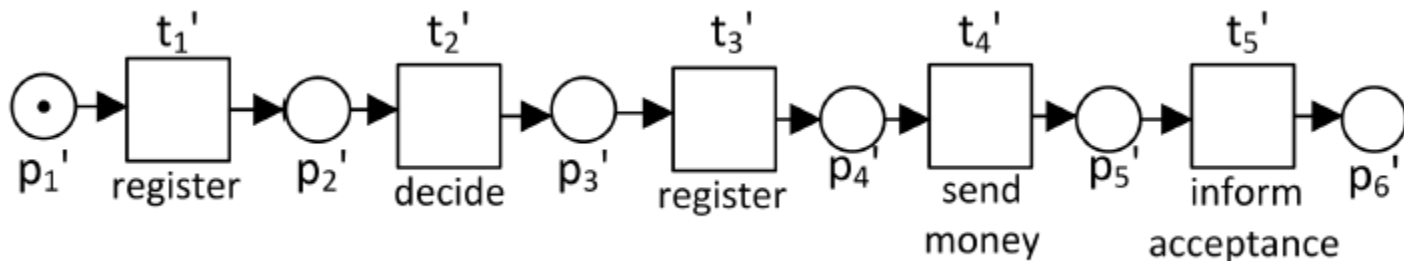
- **Goal:** given a log with traces that contains uncertainty and a (non uncertain) model, calculate a measure of conformance for the **best and worst case scenario**
 - Search among possible realization of the uncertain trace the best and worst fitting
 - Provide an upper and lower bound for conformity cost in uncertain setting
 - We are going to use **alignments**
- **Setting:**
 - Strong uncertainty on activities and timestamps
 - Strongly uncertain indeterminate events

Alignments

To align a trace with a model, we need to firstly turn the trace into an **event net**, a sequence-shaped Petri net able to execute only that specific trace.

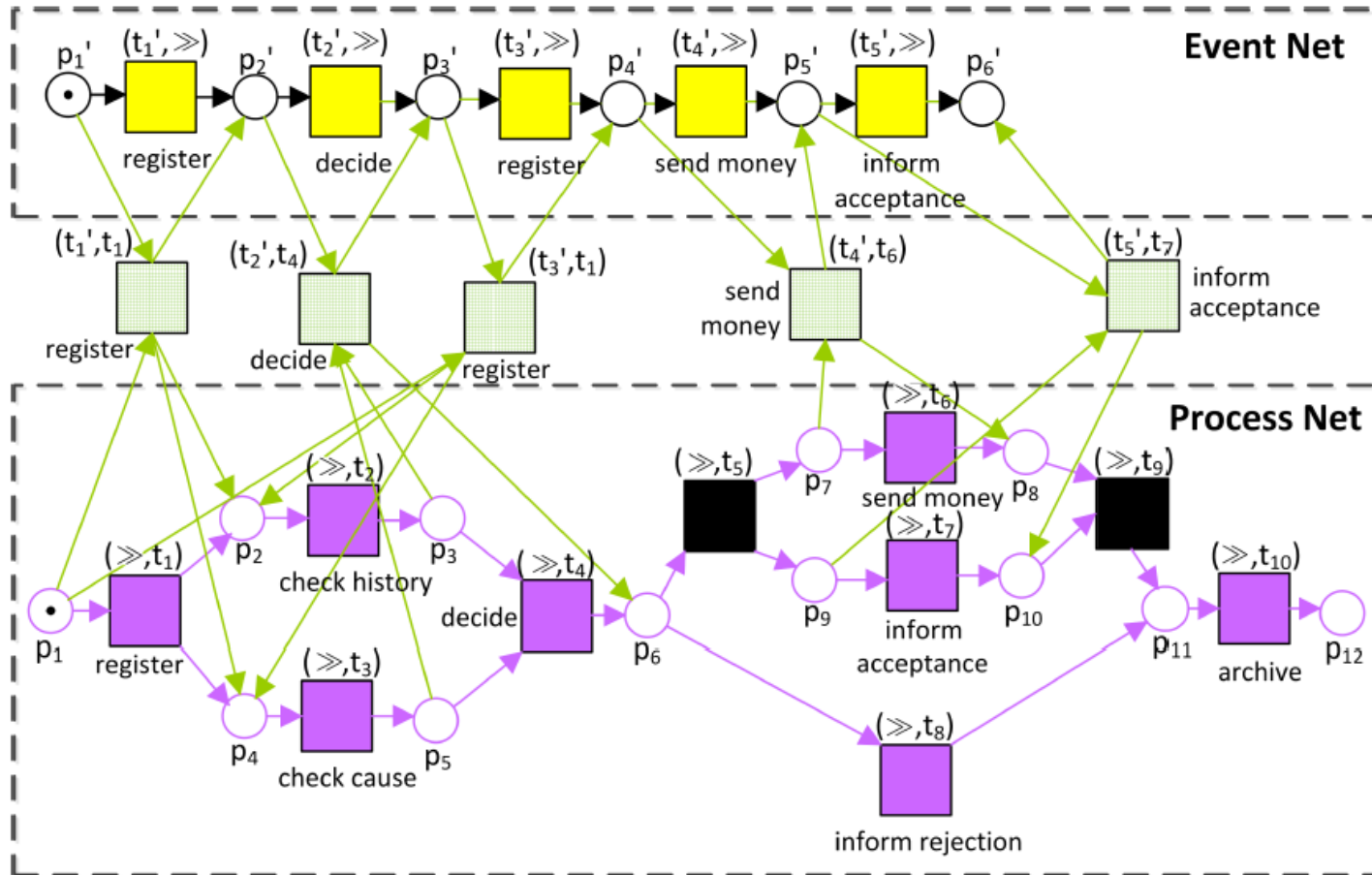


A process model



Event net of the trace *<register, decide, register, send money, inform acceptance>*

Product Net



LEGEND

Move on model

Move on model (invisible transitions)

Synchronous move

Move on log

Alignments

Transitions in a product net are labeled with **costs**.

- Synchronous moves: cost 0
- Moves on model and moves on log: cost 1

To obtain an optimal alignment from the product net, we search a complete firing sequence of the product net such that **the cost is minimal**.

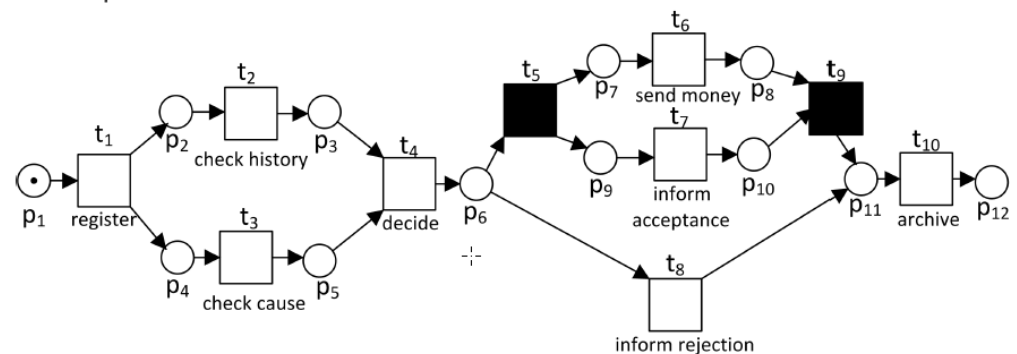
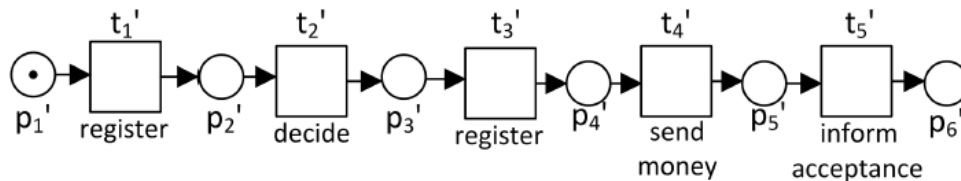
<i>register</i>	»	»	<i>decide</i>	<i>register</i>	»	<i>send money</i>	<i>inform acceptance</i>	»	»
<i>register</i>	<i>check history</i>	<i>check cause</i>	<i>decide</i>			<i>send money</i>	<i>inform acceptance</i>		<i>archive</i>
<i>t₁</i>	<i>t₂</i>	<i>t₃</i>	<i>t₄</i>	»	<i>t₅</i>	<i>t₆</i>	<i>t₇</i>	<i>t₉</i>	<i>t₁₀</i>

An optimal alignment for the trace and the model.

Alignments

<i>register</i>	\gg	\gg	<i>decide</i>	<i>register</i>	\gg	<i>send money</i>	<i>inform acceptance</i>	\gg	\gg
<i>register</i>	<i>check history</i>	<i>check cause</i>	<i>decide</i>			<i>send money</i>	<i>inform acceptance</i>		<i>archive</i>
t_1	t_2	t_3	t_4	\gg	t_5	t_6	t_7	t_9	t_{10}

Once removed the skips, the top row corresponds to the **trace**, the bottom row corresponds to a **complete firing sequence** of the model.



Running example

Setting:

- Strong uncertainty on activities and timestamps
- Strongly uncertain indeterminate events

Case ID	Timestamp	Activity	Indet. event
0	2011-12-05T00:00	A	!
0	2011-12-07T00:00	{B, C}	!
0	[2011-12-06T00:00 2011-12-10T00:00]	D	!
0	2011-12-09T00:00	{A, C}	!
0	2011-12-11T00:00	E	?

Realizations of a trace

- **Realizations of a trace:** all possible certain traces obtained by selecting an available value for the uncertain attributes.

Case ID	Timestamp	Activity	Indet. event
0	2011-12-05T00:00	A	!
0	2011-12-07T00:00	{B, C}	!
0	[2011-12-06T00:00 2011-12-10T00:00]	D	!
0	2011-12-09T00:00	{A, C}	!
0	2011-12-11T00:00	E	?

Realizations:

<A, B, C, D, E>

<A, B, D, C, E>

<A, C, D, C, E>

<A, C, D, A, E>

<A, D, C, C, E>

<A, D, B, C, E>

<A, D, C, A, E>

...

Conformance Checking in Uncertain Settings

Bruteforce approach

1. Generate all the realizations of an uncertain trace
2. Align all of them
3. Pick the ones with the minimum and maximum score

Very slow!

Can we optimize the procedure?

There is a quicker way to compute the **lower bound for conformance cost**

Conformance Checking in Uncertain Settings

Idea: **we can create a Petri net that can replay all and only the realizations of an uncertain trace**

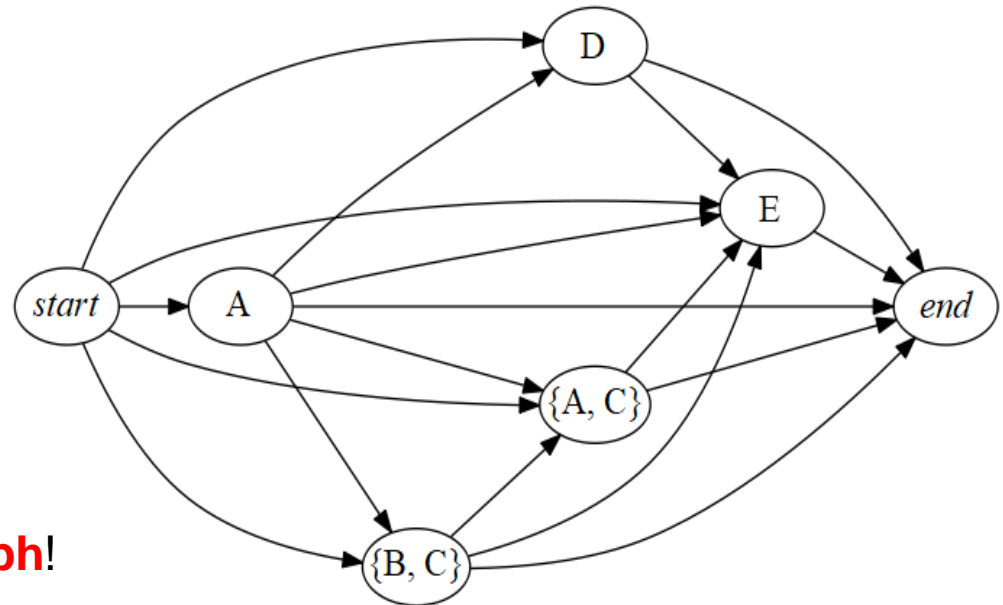
Transformation steps:

1. Create a **directed graph** that contains the information regarding timestamp uncertainty
2. Simplify the graph removing unnecessary edges
3. Transform the graph in a Petri net

Process Mining over Uncertainty: behavior graph

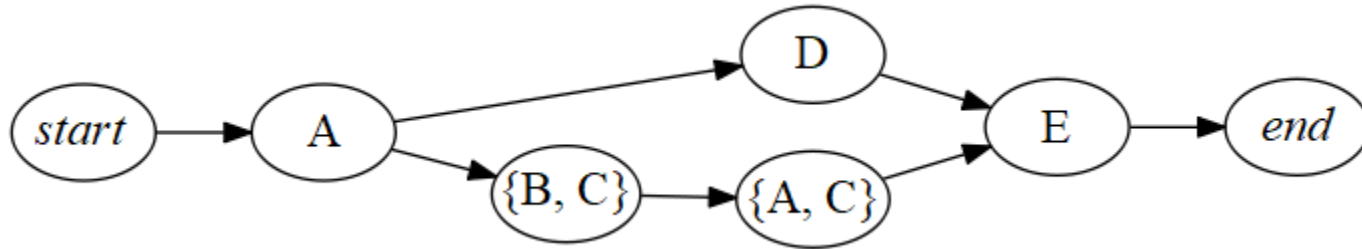
1. Create a node for each uncertain event
2. Create two extra nodes *start* and *end*
3. Connect node A with node B if the event in A **has happened** before the event in B (no overlapping)

Case ID	Timestamp	Activity	Indet. event
0	2011-12-05T00:00	A	!
0	2011-12-07T00:00	{B, C}	!
0	[2011-12-06T00:00 2011-12-10T00:00]	D	!
0	2011-12-09T00:00	{A, C}	!
0	2011-12-11T00:00	E	?



Notice that a behaviour graph will always be a **directed acyclic graph**!

Process Mining over Uncertainty: reduced behavior graph



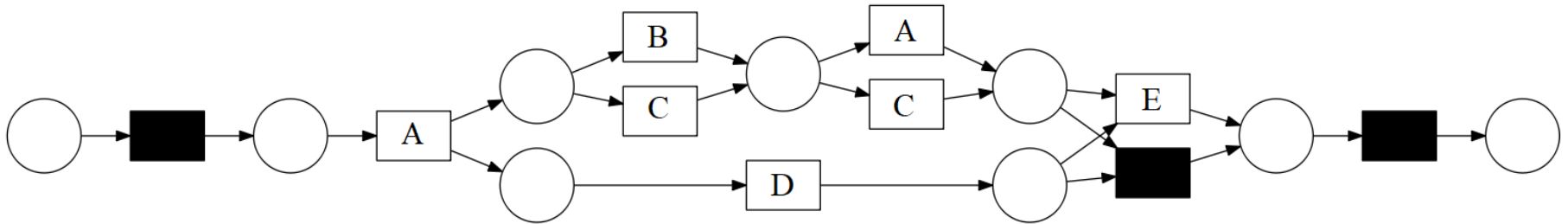
We then perform a **transitive reduction** to obtain a **reduced behavior graph**.

Transitive reduction: remove arcs until the number of arcs is **minimal** while **maintaining the connectivity** of the graph.

In DAGs, the transitive reduction **always exists and is unique** (corresponds to minimum spanning tree).

Now, the node A is connected to a node B if the event in A **could have happened immediately before** the event in B (with no events inbetween).

Process Mining over Uncertainty: behavior net

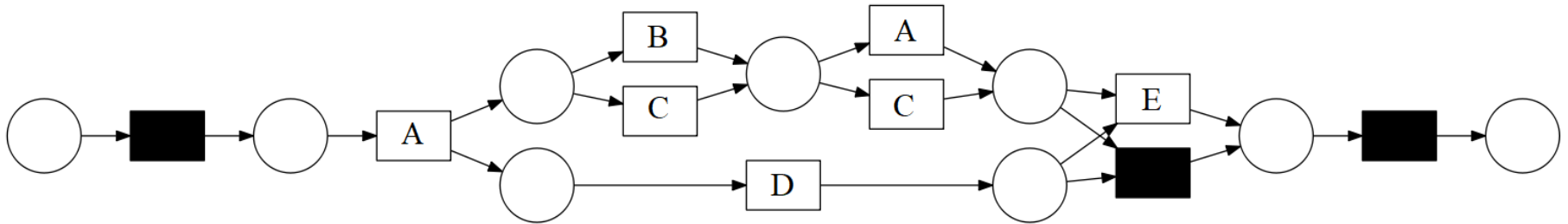


The behavior net of an uncertain trace is a system net that can execute **all and only** the realizations of the trace.

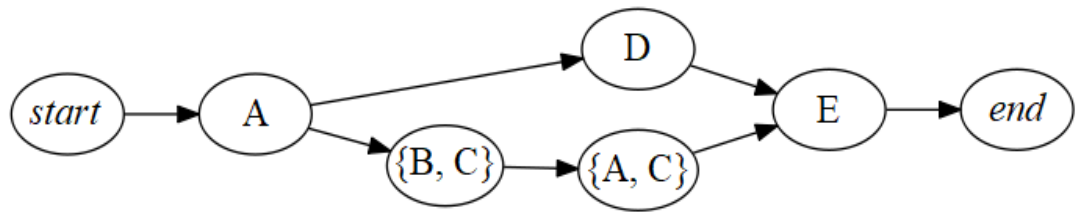
To obtain it from the reduced behavior graph:

1. Split every arc into two arcs interleaved with a **place**
2. Turn every node into a set of **transitions** adding arcs such that place connectivity is maintained
3. If an event is indeterminate, XOR it with an **invisible transition**

Process Mining over Uncertainty: behavior net



1. Split every arc into two arcs interleaved with a **place**



2. Turn every node into a set of **transitions** adding arcs such that place connectivity is maintained
3. If an event is indeterminate, XOR it with an **invisible transition**

Case ID	Timestamp	Activity	Indet. event
0	2011-12-05T00:00	A	!
0	2011-12-07T00:00	{B, C}	!
0	[2011-12-06T00:00 2011-12-10T00:00]	D	!
0	2011-12-09T00:00	{A, C}	!
0	2011-12-11T00:00	E	?

Process Mining over Uncertainty: behavior net

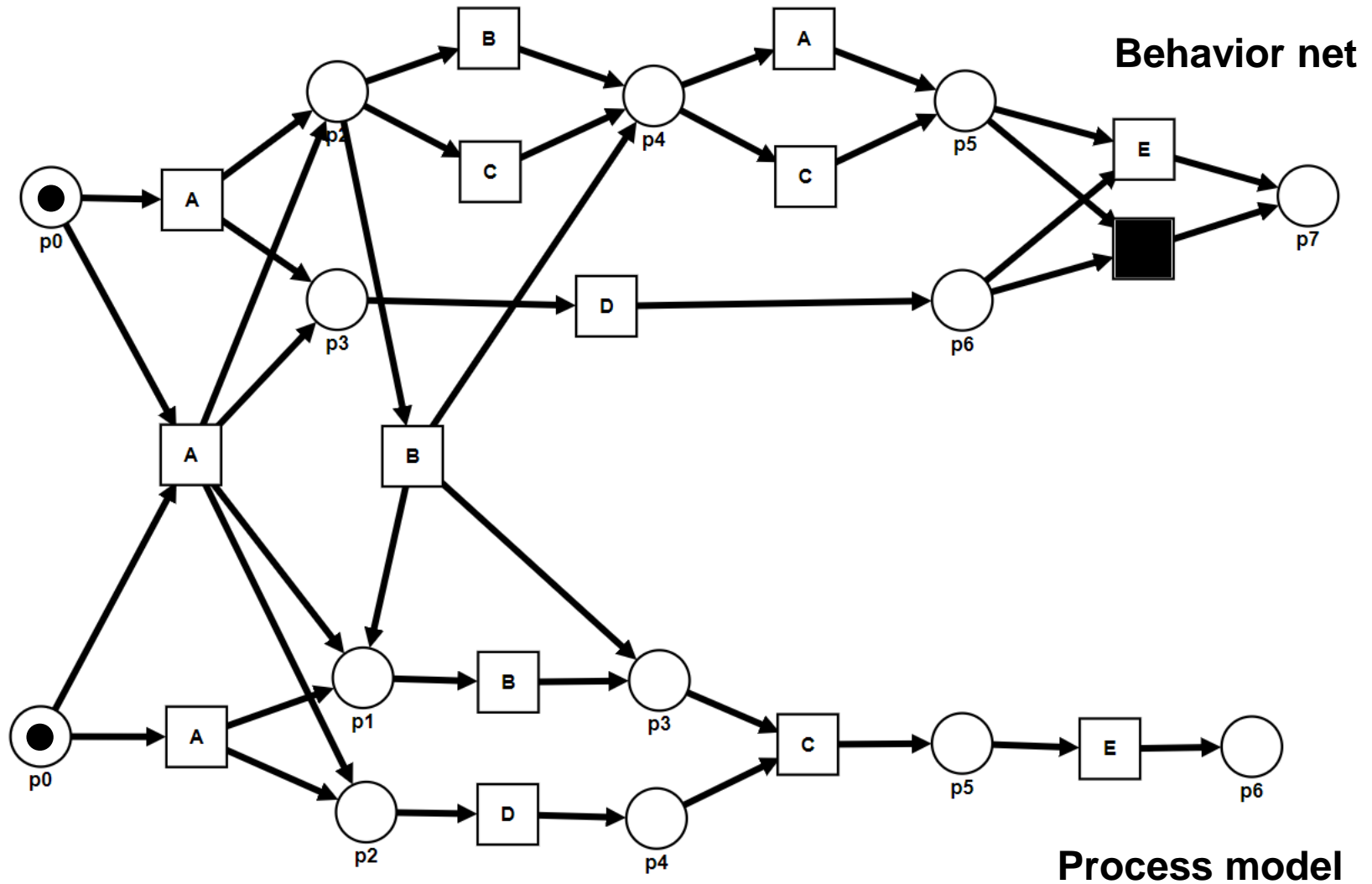
We can use the behavior net instead of the event net to compute alignments.

In this case, an alignment will be **two complete firing sequences**, one on the behavior net and one on the model.

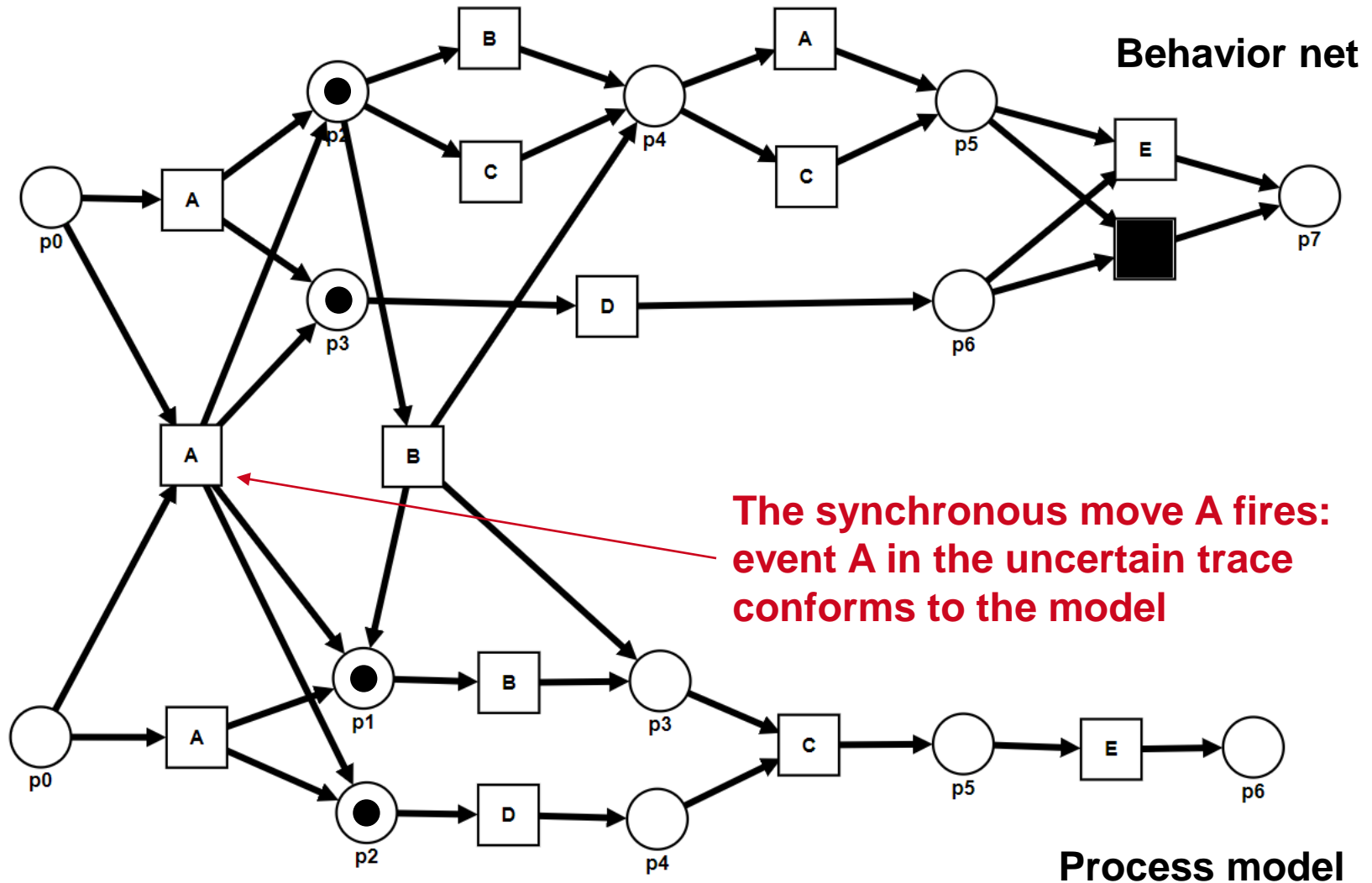
The firing sequence in the behavior net will be a **realization** of the uncertain trace, since the behavior net can replay all and only the realizations of the trace.

Since the search returns the path through the product net with the minimal cost, the realization returned by the alignment will be the **lower bound** for conformance cost

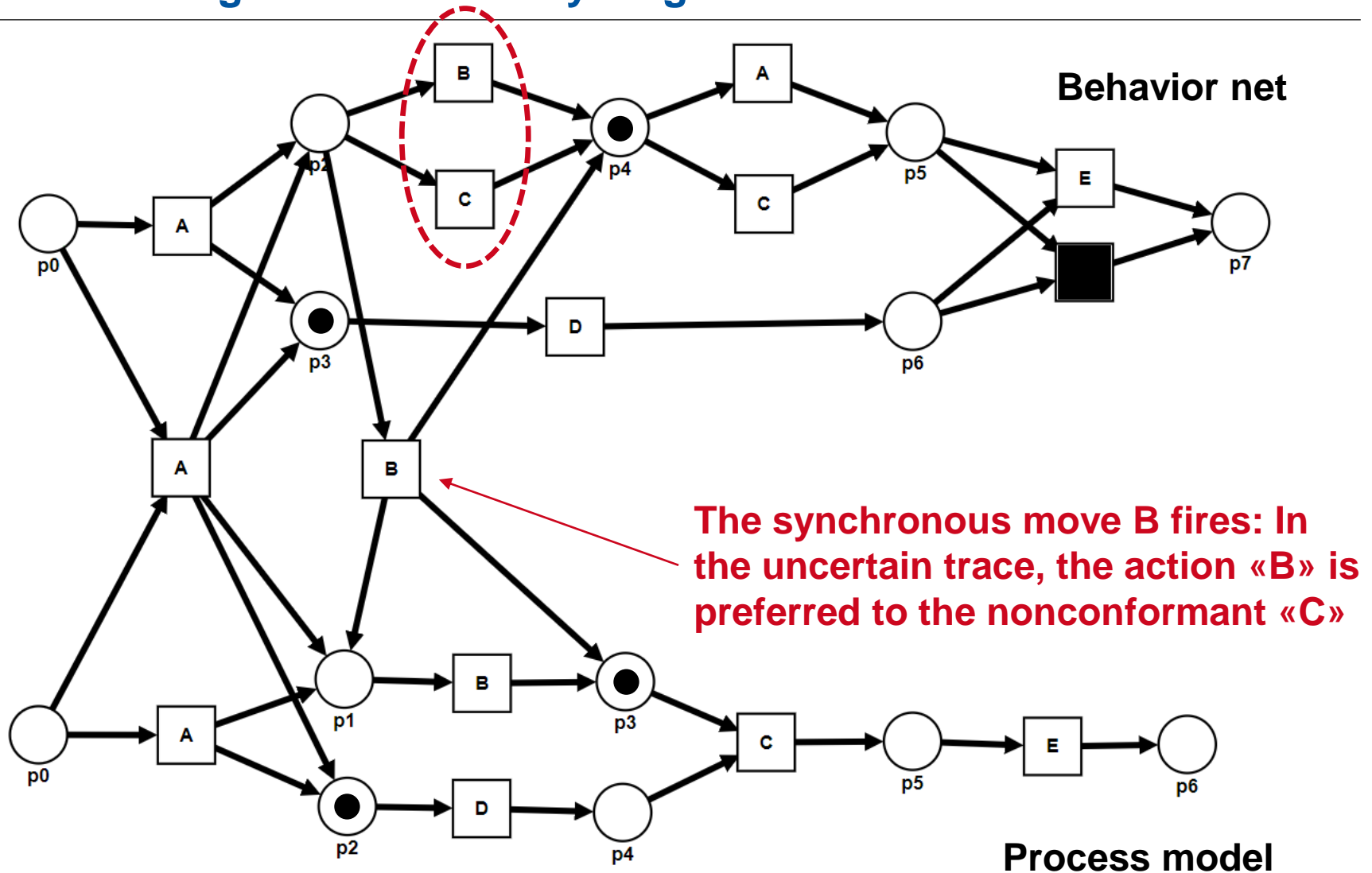
Process Mining over Uncertainty: alignments



Process Mining over Uncertainty: alignments



Process Mining over Uncertainty: alignments



Experimental results

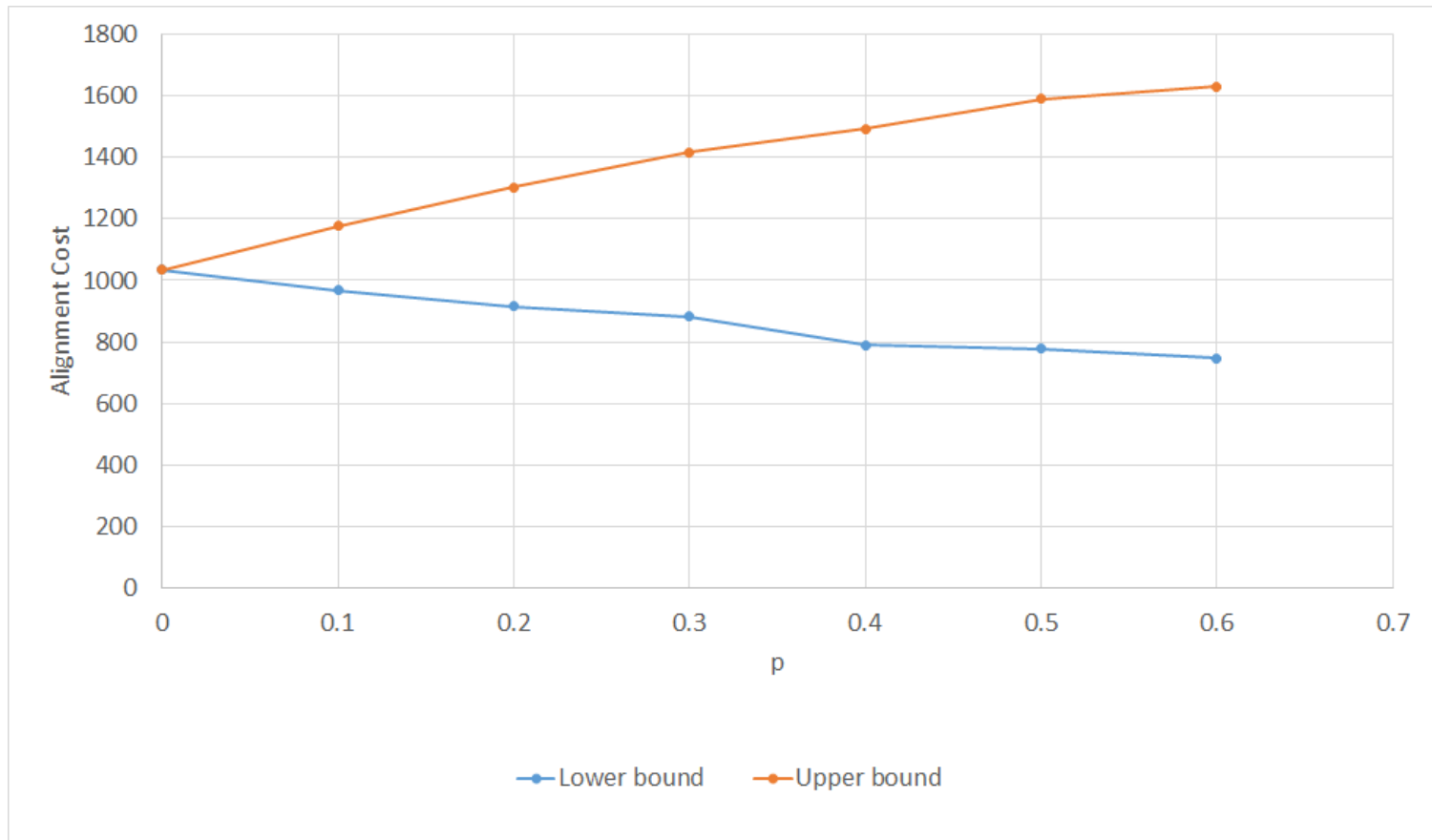
Performed experiments to answer two research questions:

- **Q1: do the upper and lower bounds for the conformance cost behave as expected in uncertain traces?**
- **Q2: does aligning the behavior net to calculate the lower bound for the conformance cost yield lower computing times?**

Experimental setting Q1

1. Generate a model
2. Generate an event log from the model
3. Introduce deviations in the event log
4. With increasing values of p
 1. Create a copy of the log with uncertainty in the events with probability p
 2. Compute upper and lower bound for conformance cost

Experimental results: Q1

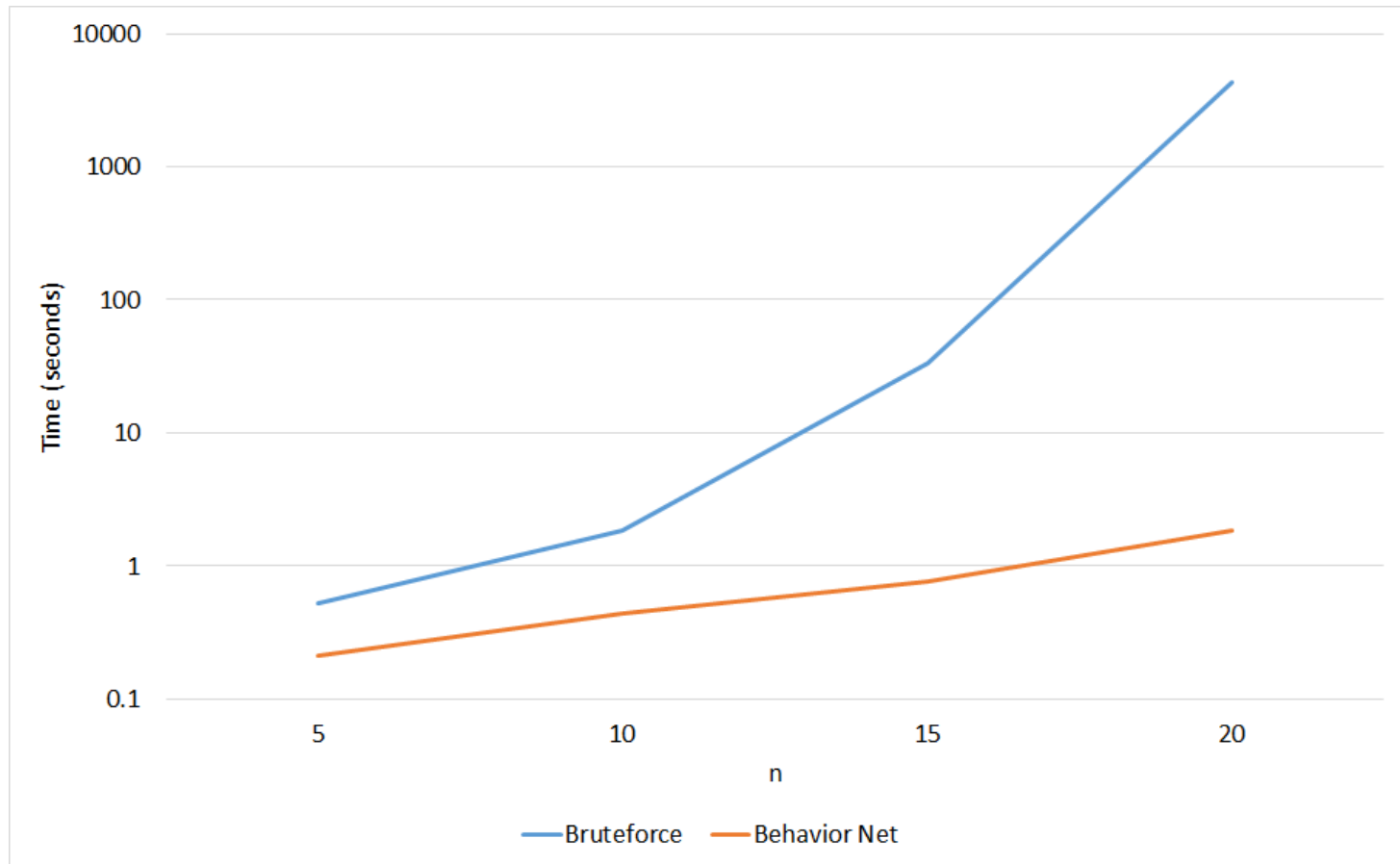


The change in lower and upper bound for conformance checking of a noisy event log of 250 traces with increasing probability of having uncertainty on event data. p is the probability of having uncertainty in a trace.

Experimental setting Q2

1. With an increasing number of transitions
 1. Generate a model
 2. Generate an event log from the model
 3. Compute the lower bound for conformance (bruteforce)
 4. Compute the lower bound for conformance (behavior net)
 5. Compare the cost in CPU time

Experimental results: Q2

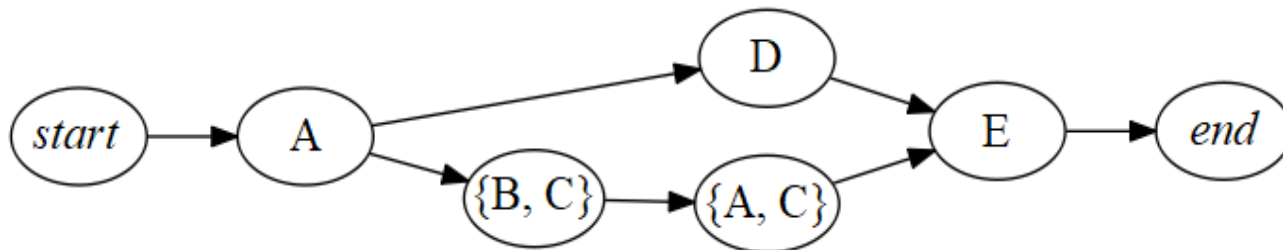


Effect on time performance of calculating the lower bound for conformance cost with the bruteforce method vs. the behavior net. n is the number of transitions. For $n=20$, the behavior net takes 0.04% of the time needed by the bruteforce (x2000 speedup).

- Preliminaries and Definitions
- Conformance Checking over Uncertainty
- Behavior Graph Construction
- Conclusion

Behavior graph

Case ID	Timestamp	Activity
0	2011-12-05T00:00	A
0	2011-12-07T00:00	{B, C}
0	[2011-12-06T00:00 2011-12-10T00:00]	D
0	2011-12-09T00:00	{A, C}
0	2011-12-11T00:00	E



An event e_1 is connected to an event e_2 if and only if e_2 **may directly follow** e_1 and the timestamps of the two events **do not overlap**

Behavior graph creation

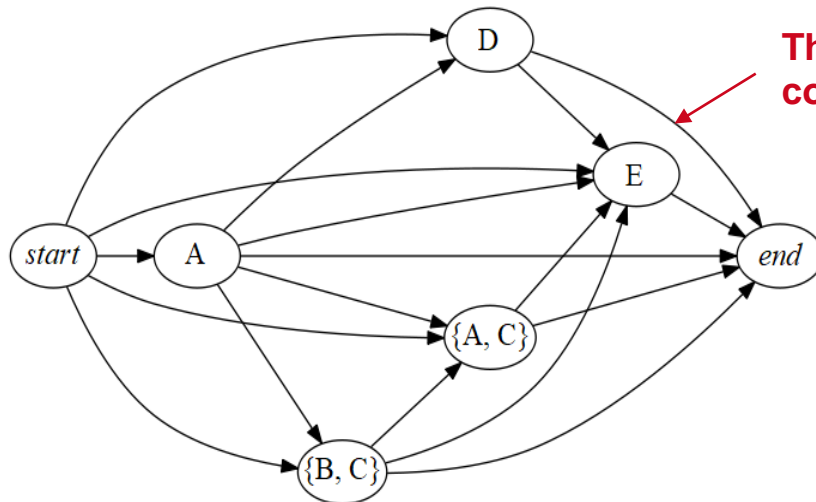
Conceptually simple to obtain:

- 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

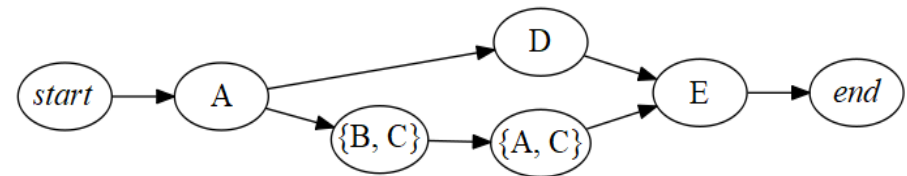
Transitive reduction: removing the maximum number of edges from a graph without altering the **reachability** between nodes.

Behavior graph creation

Case ID	Timestamp	Activity
0	2011-12-05T00:00	A
0	2011-12-07T00:00	{B, C}
0	[2011-12-06T00:00 2011-12-10T00:00]	D
0	2011-12-09T00:00	{A, C}
0	2011-12-11T00:00	E



This edge is removed, D is connected to end through E



Behavior graph creation

- This procedure is useful to explain the function of the behavior graph from a formal standpoint
- But... It is expensive!

Behavior graph creation

- This procedure is useful to explain the function of the behavior graph from a formal standpoint
- But... It is expensive!
- Consider an uncertain trace of length **n**
 - Construction of the starting graph: we have to compare all pairs of events, so n^2 operations
 - The resulting graph is a generic DAG, so it can have up to $\frac{n(n-1)}{2}$ edges

Behavior graph creation

- This procedure is useful to explain the function of the behavior graph from a formal standpoint
- But... It is expensive!
- Consider an uncertain trace of length **n**
 - Construction of the starting graph: we have to compare all pairs of events, so n^2 operations
 - The resulting graph is a generic DAG, so it can have up to $\frac{n(n-1)}{2}$ edges
 - The transitive reduction is performed in cubic time on nodes

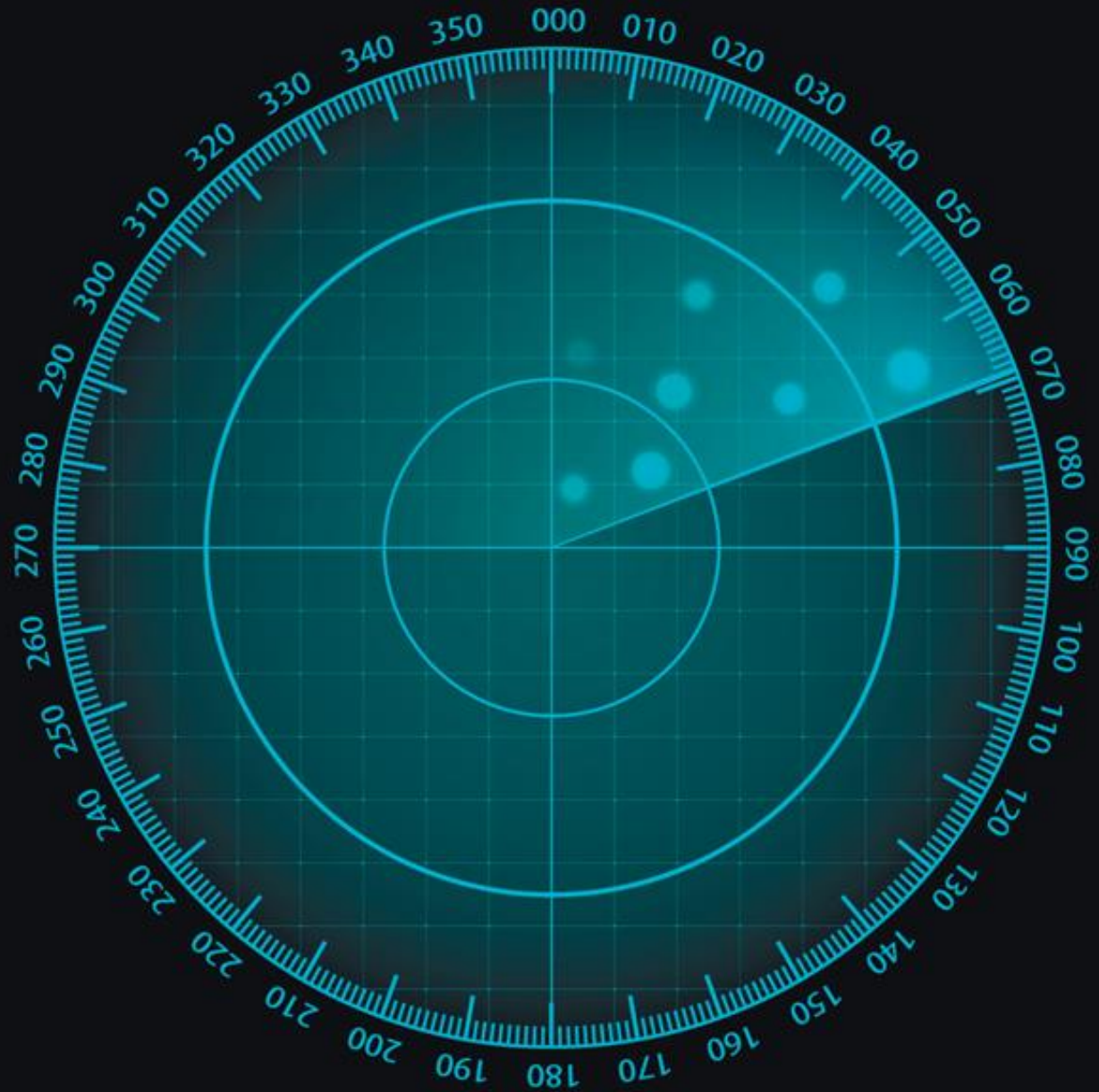
Behavior graph creation

- This procedure is useful to explain the function of the behavior graph from a formal standpoint
- But... It is expensive!
- Consider an uncertain trace of length n
 - Construction of the starting graph: we have to compare all pairs of events, so n^2 operations
 - The resulting graph is a generic DAG, so it can have up to $\frac{n(n-1)}{2}$ edges
 - The transitive reduction is performed in cubic time on nodes
 - Total cost: $O(n^2) + O(n^3) = O(n^3)$

Behavior graph creation

- Why creating behavior graphs through transitive reduction is inefficient?
- Intuitively, it is because **we are adding** (a lot of) **edges** that we then **delete later**!
- Is it possible to better determine the edges in a behavior graph?

Idea:



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

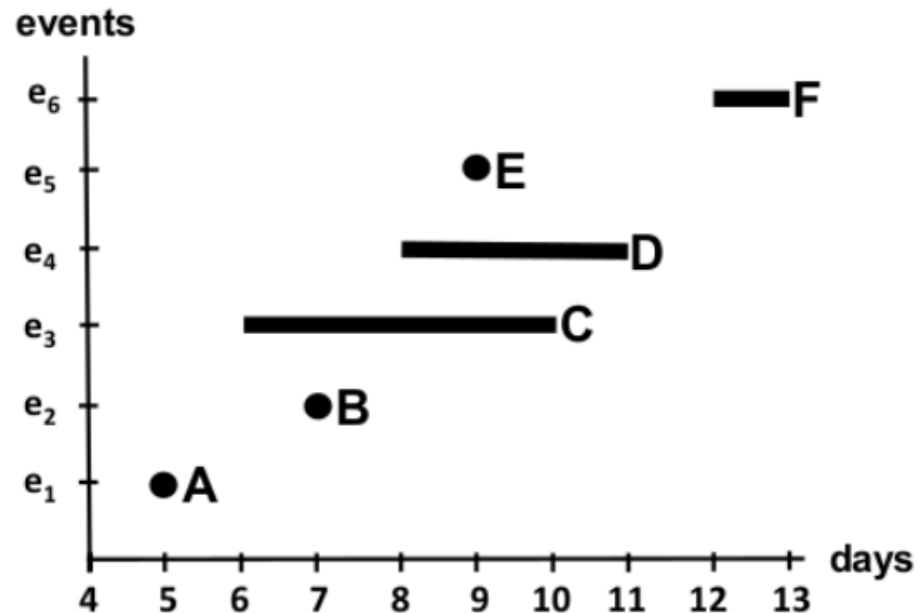
- Preprocessing step: transform the uncertain trace in a list of activities sorted by timestamp
- In this list, events with an uncertain timestamps become **two entries**: the left and right extreme of the time interval are inserted in the list, labeled as such, with a reference to the original activity 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 dimension 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

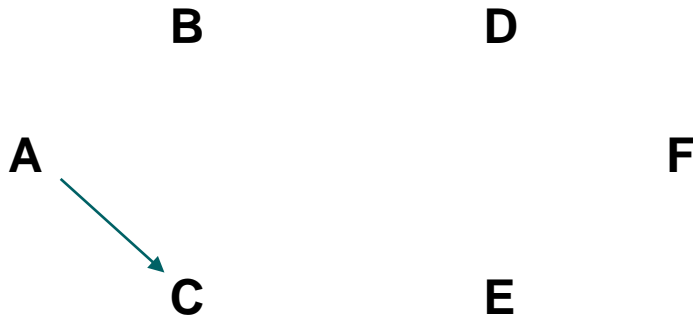
```

A diagram showing six points labeled A, B, C, D, E, and F arranged in a regular hexagon. The points are connected by lines forming the hexagon's perimeter.

Behavior graph creation: example

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

↑ ↑

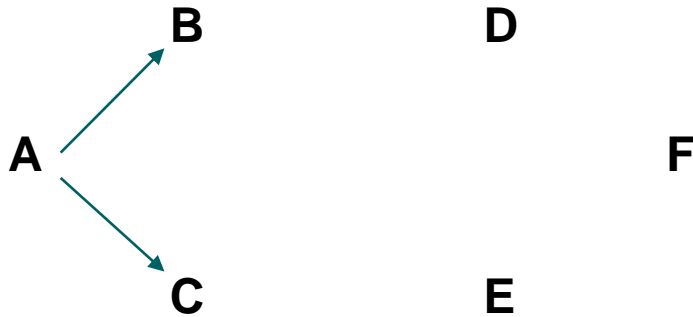


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

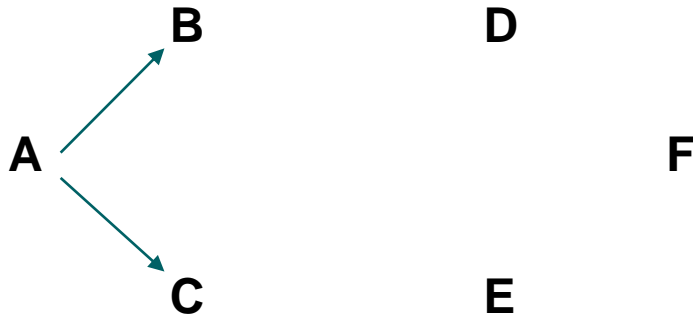
↑ ↑



```
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

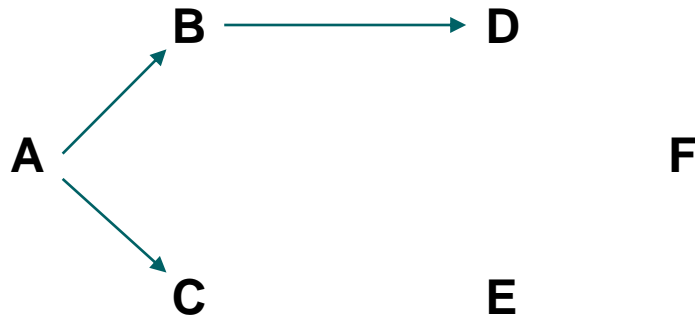


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

↑ ↑

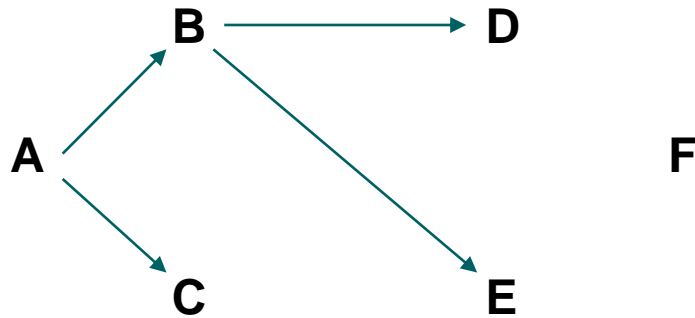


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

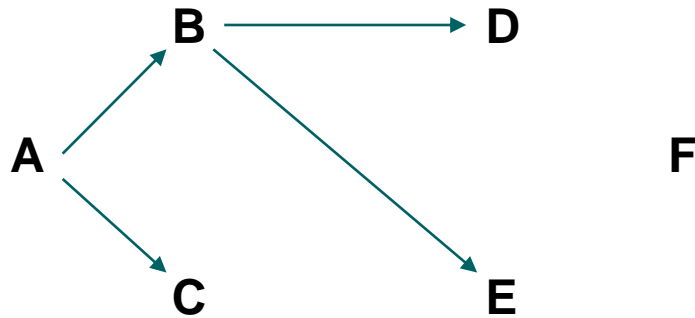
↑ ↑



```
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

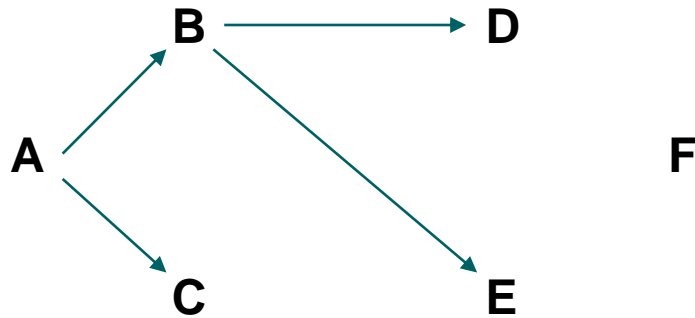


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

↑ ↑

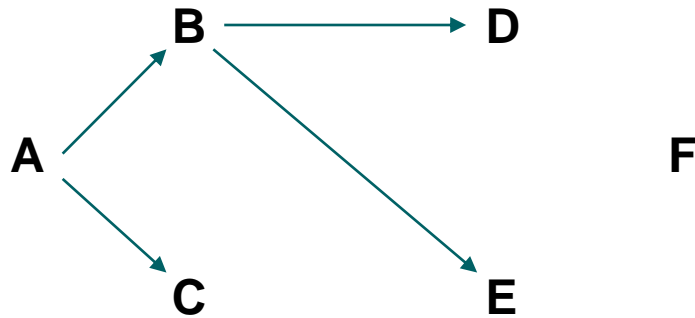


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

↑ ↑

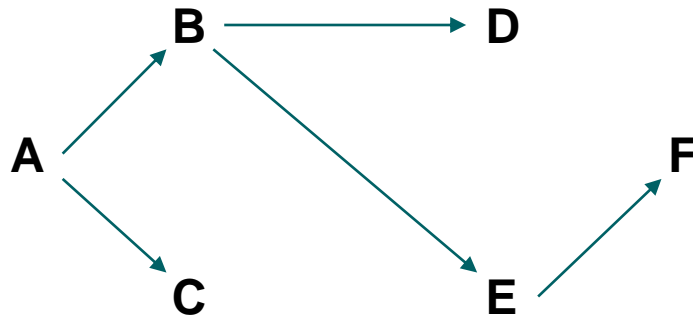


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

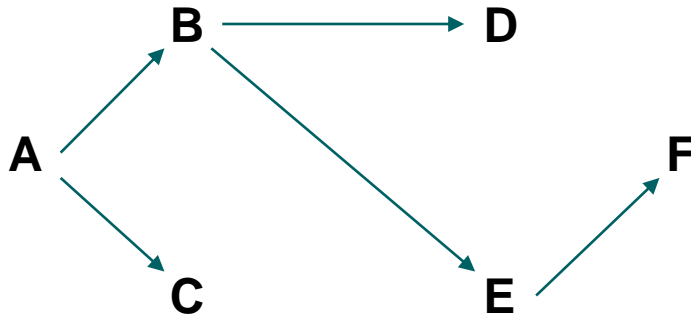
↑ ↑



```
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



```

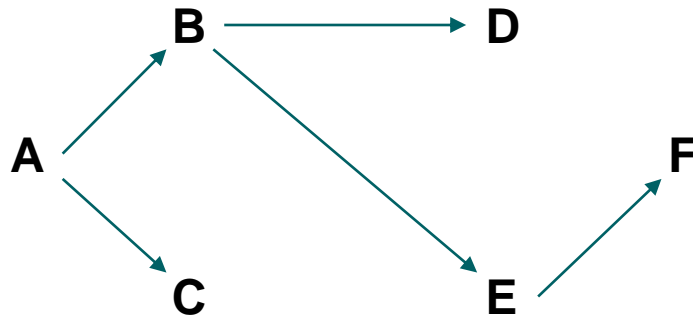
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme
  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

↑ ↑

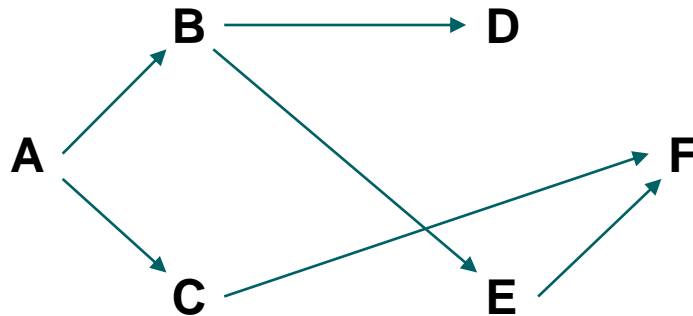


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

Behavior graph creation: example

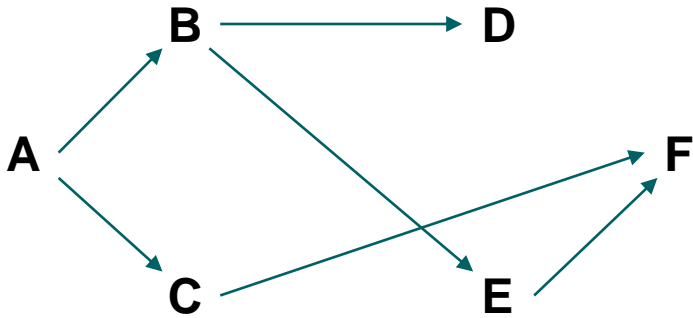
A, C_L, B, D_L, E, C_R, D_R, F_L, F_R

↑ ↑



```
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
```


↑ ↑



```

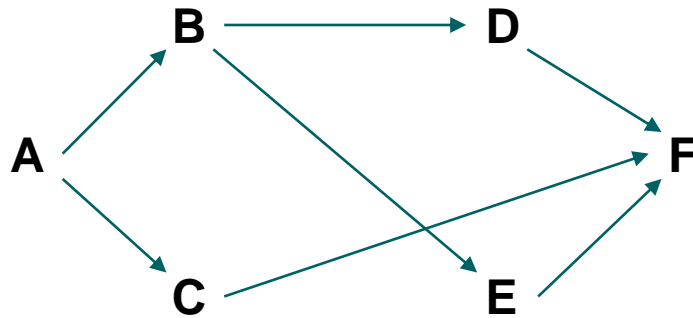
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme
  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

↑ ↑

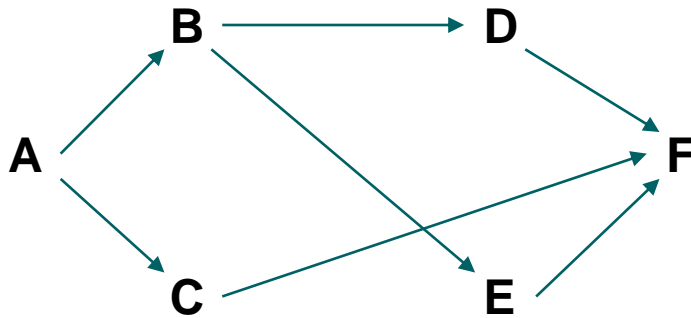


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

↑ ↑

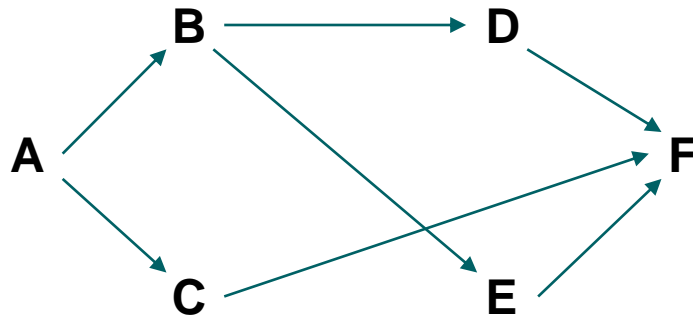


```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

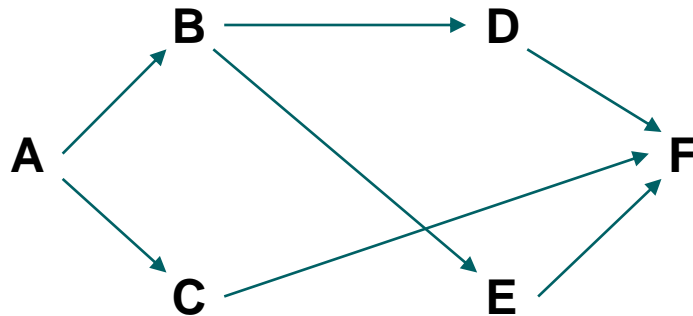
↑



```
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

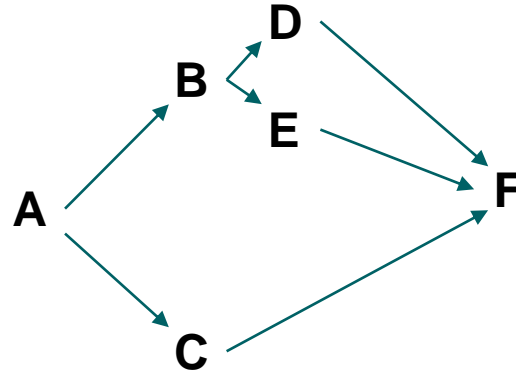
A, C_L, B, D_L, E, C_R, D_R, F_L, **F_R**



```
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: correctness

For connections, we consider only certain timestamps and right extremes of intervals because an uncertain node can have an outbound arc only towards events that **surely** happened afterwards.

```
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: correctness

When we encounter the left extreme of an uncertain timestamp in the search, I have to connect it with the current node – it is a successor.

However, it is not the only one: **all the events before the corresponding right extreme are potential successors** as well, so the search must continue.

```
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: correctness

When we encounter an event with a certain timestamp in the search, I have to connect it with the current node – it is a successor.

It is also the only one: it happens in a precise point in time, and everything else will happen afterwards. So the search must stop.

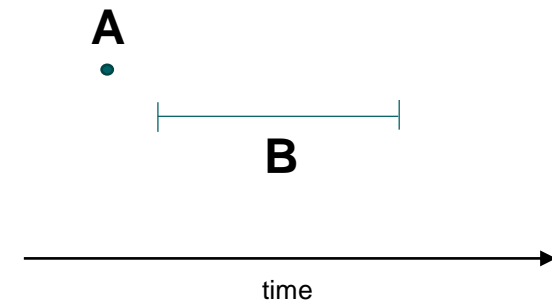
```
for i = 1 to n
  if list[i] is not a left extreme
    for j = i+1 to n
      if list[j] is a left extreme
        connect list[i] node with list[j] node and continue
      if list[j] is a certain timestamp
        connect list[i] node with list[j] node and stop
      if list[j] is a right extreme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

Behavior graph creation: correctness

When we encounter the right extreme of an uncertain timestamp in the search there are two possible cases:

If nodes were already connected, in the search I had already encountered the left extreme of the same interval, so now **I discovered an entire uncertain event** that blocks possible successors. So the search must stop.

```
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
```

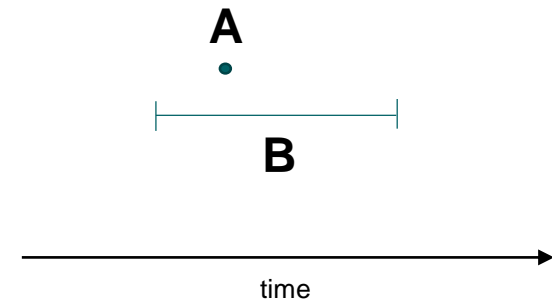


Behavior graph creation: correctness

When we encounter the right extreme of an uncertain timestamp in the search there are two possible cases:

If nodes were *not* already connected, that means that the two events have **overlapping timestamps**. Thus, we do not connect them, but the search continues.

```
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: 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 $O(2n \cdot \log(2n) + 2n \cdot 2n) = \mathbf{O(n^2)}$.

Behavior graph creation: complexity

To also find the lower bound we can exploit **some properties of behavior graphs**.

Namely, a behavior graph is a **directed acyclic graph which is equal to its transitive reduction** (it has the minimal set of edges to conserve reachability).

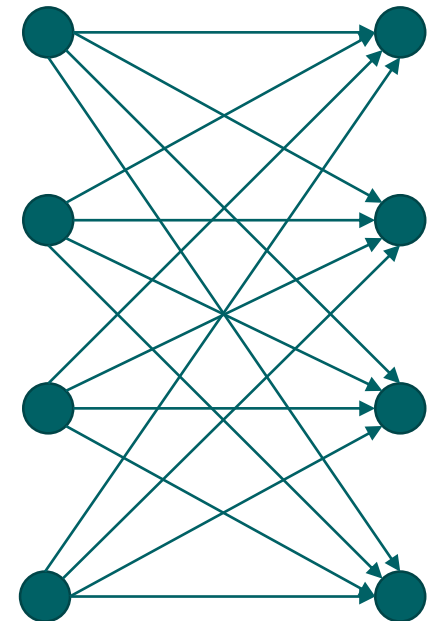
Since the algorithm builds such graphs, it has to perform a number of operations **at least linear in the size** of those graphs.

Behavior graph creation: complexity

A graph in such class is the **directed complete bipartite graph (k,k)** .

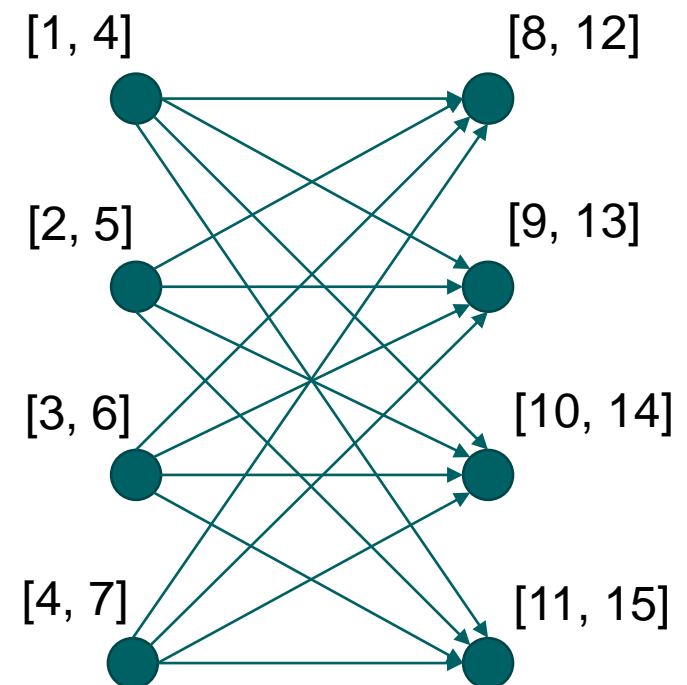
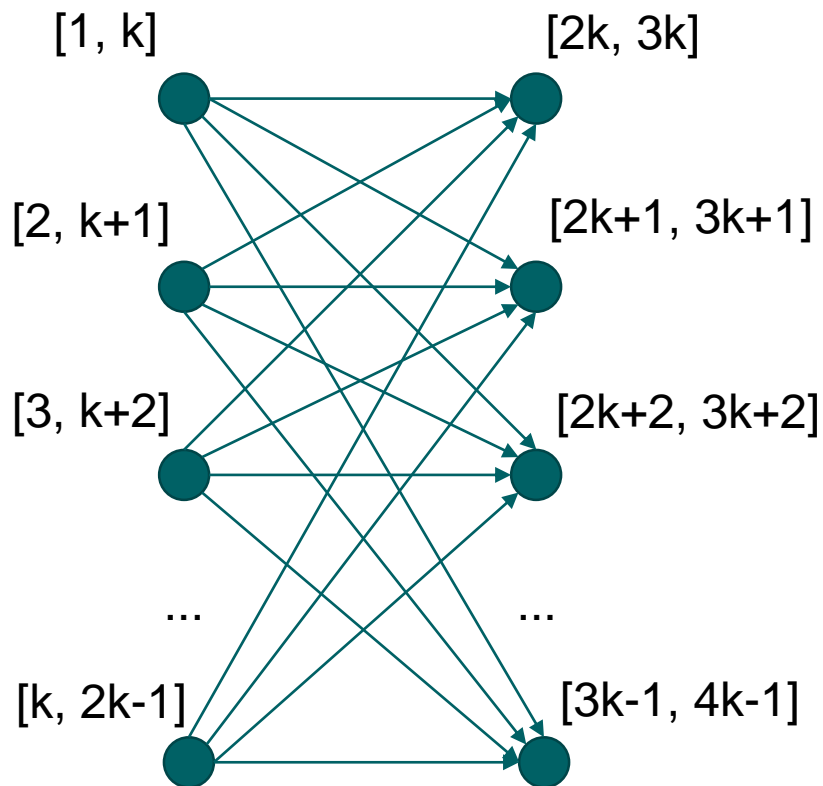
It is very easy to see that:

- This graph is **acyclic**
- **Removing any edge changes the reachability**



Behavior graph creation: complexity

There is a simple construction that proves that a behavior graph **can be homomorph to a (k,k) for an arbitrary value of k .**



Behavior graph creation: complexity

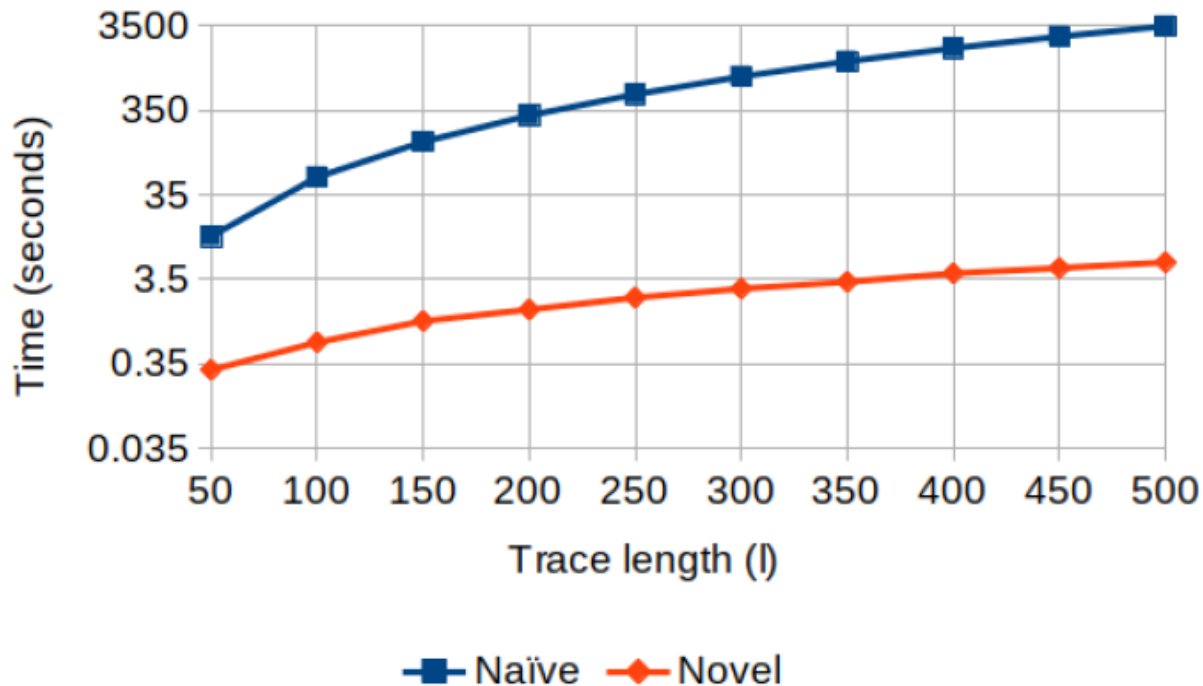
A (k,k) graph has k^2 edges.

Thus, a behavior graph homeomorph to a directed complete bipartite graph obtained from an uncertain trace of n events has $\left(\frac{n}{2}\right)^2 = O(n^2)$ edges.

This proves two assertions:

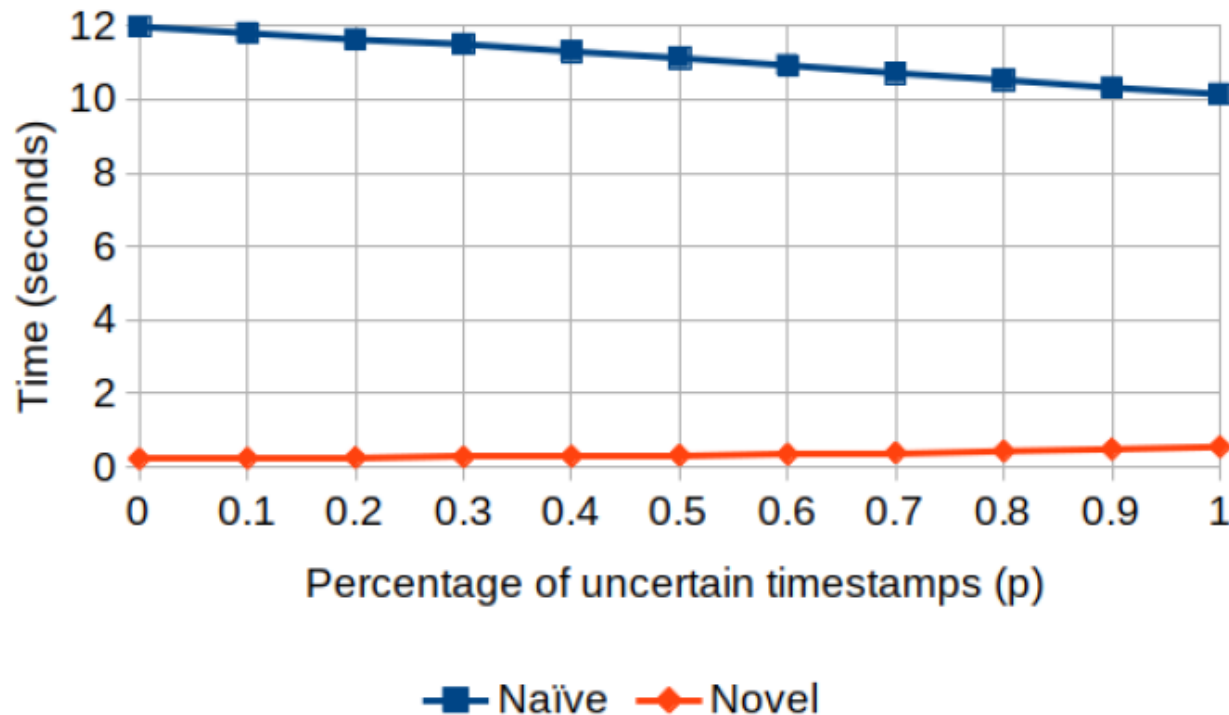
- The algorithm runs in $\omega(n^2)$, so it is also in complexity $\Theta(n^2)$
- The algorithm is **asymptotically optimal**: since a quadratic work is required to build some graphs, no algorithm can have a complexity lower than $O(n^2)$.

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 and Definitions
- Conformance Checking over Uncertainty
- Behavior Graph Construction
- Conclusion

Conclusion

- Uncertain data in process mining hinder analysis, but can be managed without deletions with dedicated techniques
- Uncertain traces contain **behavior**, and can be represented by models (graphs and/or Petri nets)
- Extensions of classic process mining techniques can provide **bounds** for the **conformance** score
- More research is needed (and is ongoing!)

Contacts and references

"Mining Uncertain Event Data in Process Mining"

Marco Pegoraro, Wil M.P. van der Aalst

International Conference on Process Mining (ICPM), 2019.

"Efficient Construction of Behavior Graphs for Uncertain Event Data"

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

International Conference on Business Information Systems (BIS), 2020.



Marco Pegoraro

pegoraro@pads.rwth-aachen.de

Site: <http://mpegoraro.net/>



<http://www.pads.rwth-aachen.de/>

Twitter: @pads_rwth

Blog: <https://blog.rwth-aachen.de/pads/>