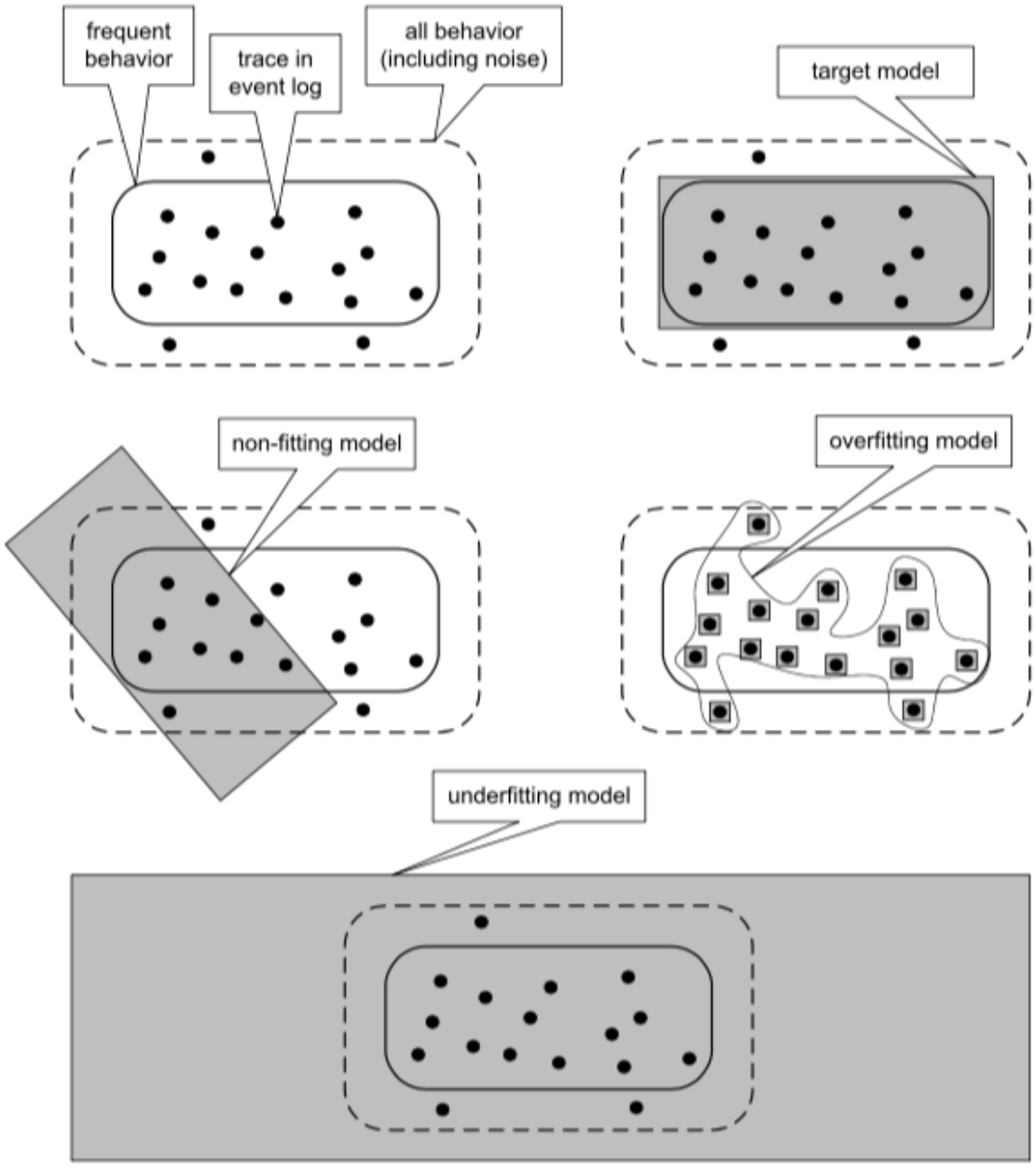
Advanced Process Discovery Techniques

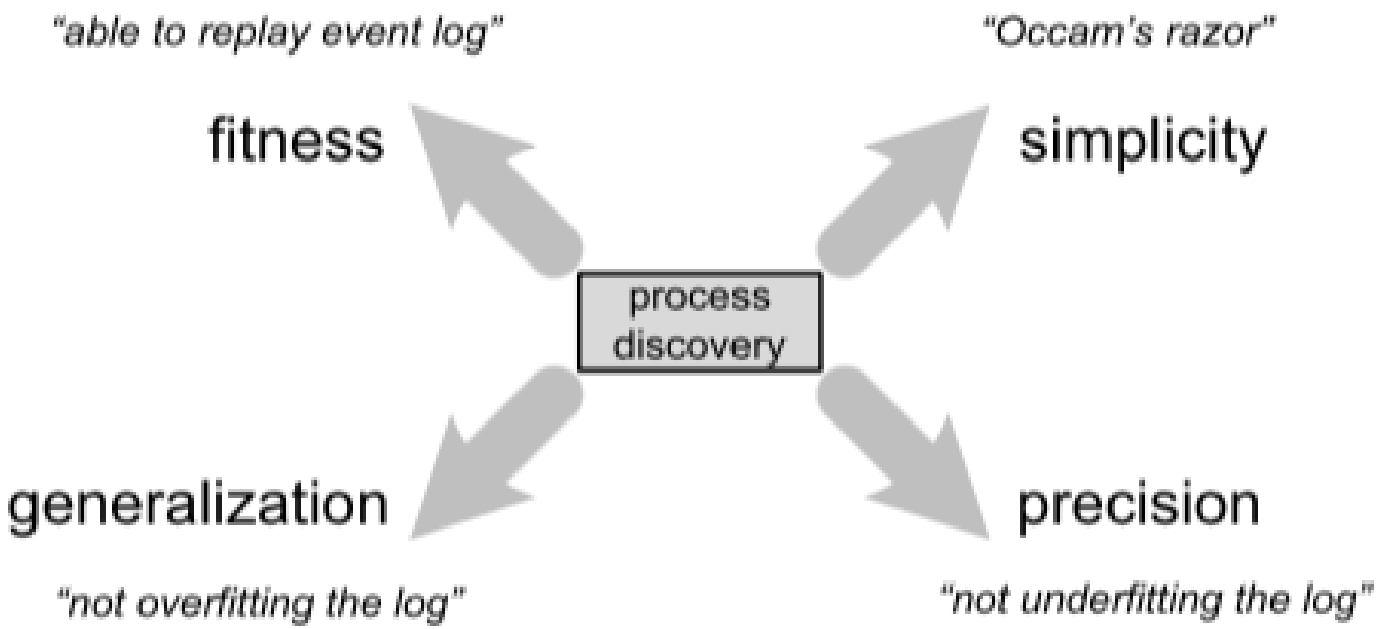
Barbara Re

Process Mining

Four Competing Quality Criteria



Quality Dinensions



Fitness

A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end

There are various ways of defining fitness.

- It can be defined at the **case level**, e.g., the fraction of traces in the log that can be fully replayed.
- It can also be defined at the **event level**, e.g., the fraction of events in the log that are indeed possible according to the model.

Simplicity

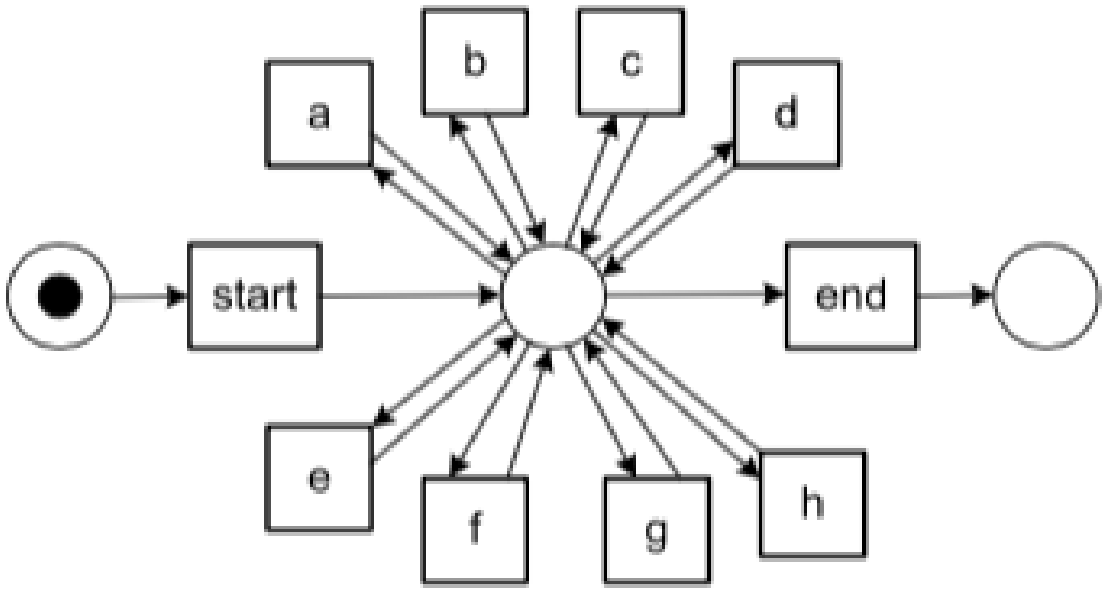
The simplicity dimension refers to Occam's Razor. In the context of process discovery this means that the simplest model that can explain the behavior seen in the log, is the best model.

The principle states that one should not increase, beyond what is necessary, the number of entities required to explain anything, i.e., one should look for the simplest model that can explain what is observed in the data set. This principle is related to finding a natural balance between overfitting and underfitting.

- The complexity of the model could be defined by the number of nodes and arcs in the underlying graph
- Also more sophisticated metrics can be used, e.g., metrics that take the **structuredness** or **entropy** of the model into account.

Simplicity

Fitness and simplicity alone are not adequate!



The flower Petri net allows for any sequence starting with start and ending with end and containing any ordering of activities in between

This model allows for all event logs used to introduce the α-algorithm., the added start and end activities are just a technicality to turn the **flower model** into a WF-net

From flower model to enumerating model

The flower model does not contain any knowledge other than the activities in the log

The flower model can be constructed based on the occurrences of activities only, and the resulting model is simple and has a perfect fitness

Considering fitness and simplicity the flower model is acceptable, this shows that they are necessary, but not sufficient

If the flower model is on one end of the spectrum, then the enumerating model is on the other end of the spectrum

The enumerating model of a log simply lists all the sequences possible, i.e., there is a separate sequential process fragment for each trace in the model

- At the start there is one big XOR split selecting one of the sequences and at the end these sequences are joined using one big XOR join
- If such a model is represented by a Petri net and all traces are unique, then the number of transitions is equal to the number of events in the log.

The enumerating model is simply an encoding of the log. Such a model is complex but, like the flower model, has a perfect fitness.

From flower model to enumerating model

Extreme models:

- the flower model (anything is possible)
- the enumerating model (only the log is possible)

show the need for two additional dimensions!!

Precision and Generalization

A model is precise if it does not allow for too much behavior

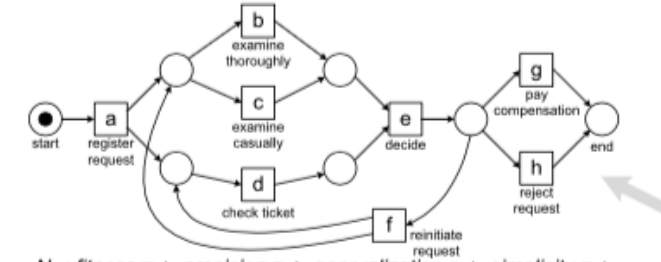
- The flower model lacks precision
- A model that is not precise is underfitting
- Underfitting is the problem that the model over-generalizes the example behavior in the log, i.e., the model allows for behaviors very different from what was seen in the log.

A model should generalize and not restrict behavior

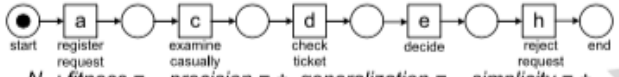
- The enumerating model lacks generalization
- A model that does not generalize is overfitting
- Overfitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior, i.e., the model explains the particular sample log, but a next sample log of the same process may produce a completely different process model.

Process mining algorithms need to strike a balance between overfitting and underfitting

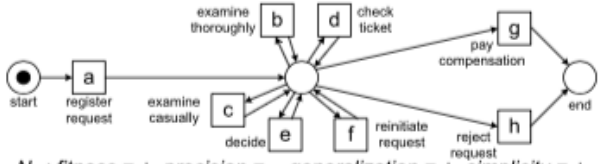
Four alternative models for the same log



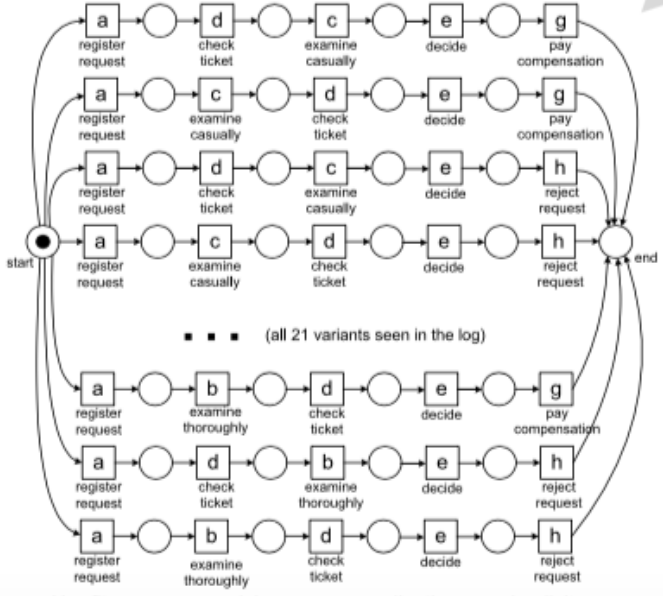
N_1 : *fitness* = +, *precision* = +, *generalization* = +, *simplicity* = +



N_2 : *fitness* = -, *precision* = +, *generalization* = -, *simplicity* = +



N_3 : *fitness* = +, *precision* = -, *generalization* = +, *simplicity* = +



N_4 : *fitness* = +, *precision* = +, *generalization* = -, *simplicity* = -

#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

Characteristics of Process Discovery Alghoritms

- Representational Bias
  - Inability to represent concurrency
  - Inability to deal with (arbitrary) loops
  - Inability to represent silent actions
  - Inability to represent duplicate actions
  - Inability to model OR-splits/joins
  - Inability to represent non-free-choice behavior
  - Inability to represent hierarchy
- Ability to Deal With Noise
- Completeness Notion Assumed
- Approach Used
  - Direct Algorithmic Approaches
  - Two-Phase Approaches
  - Divide-and-Conquer Approaches
  - Computational Intelligence Approaches
  - Partial Approaches

Heuristic Mining

Dependency Graph

Dependency graphs are very important to get the causal structure of a process model, it is often used as input for all kind of algorithms that look for more refined representations

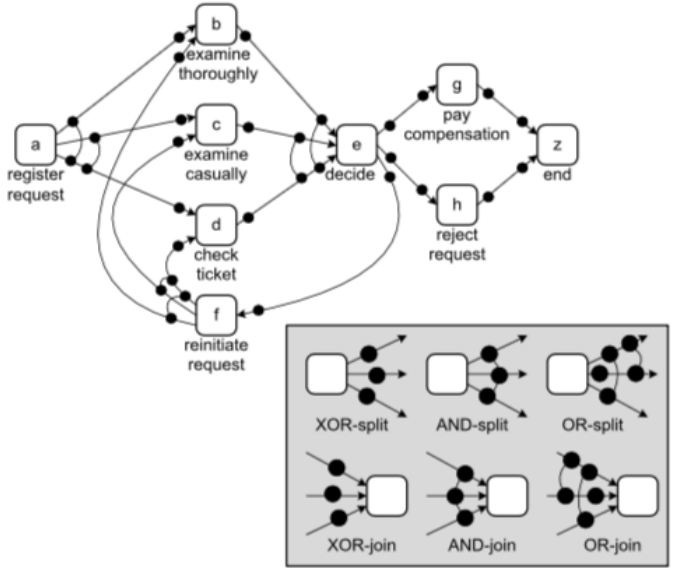
- Node correspond to activities
- Arcs correspond to casual dependencies and can be annotated with number

It has no executable semantics! Dependency graphs have a fuzzy semantics

A Petri net can be translated into a dependency graph considering the arrows in the footprint, so removing places, but we lose lots of the semantics

C-Nets

C-nets take these dependency graphs and add a bit more semantics



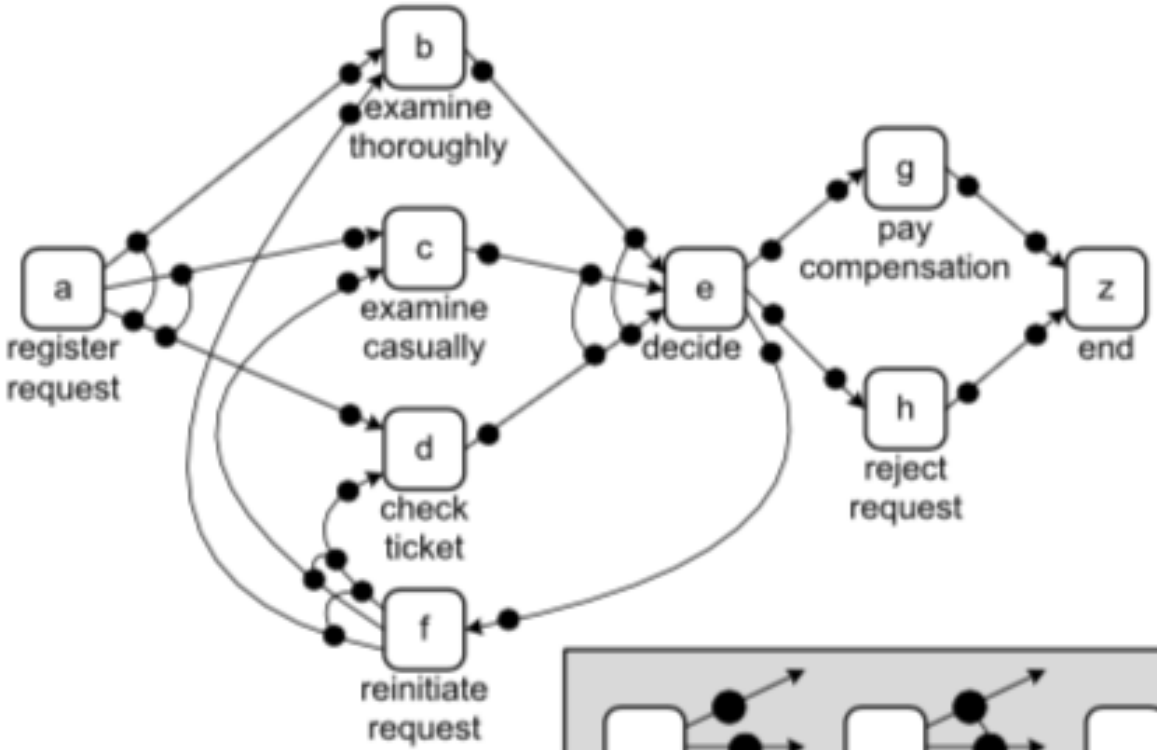
Causal net is a graph where nodes represent activities and arcs represent causal dependencies. Each activity has a set of input bindings and a set of output bindings. There are no places in the causal net and the routing logic is solely represented by the possible input and output bindings.

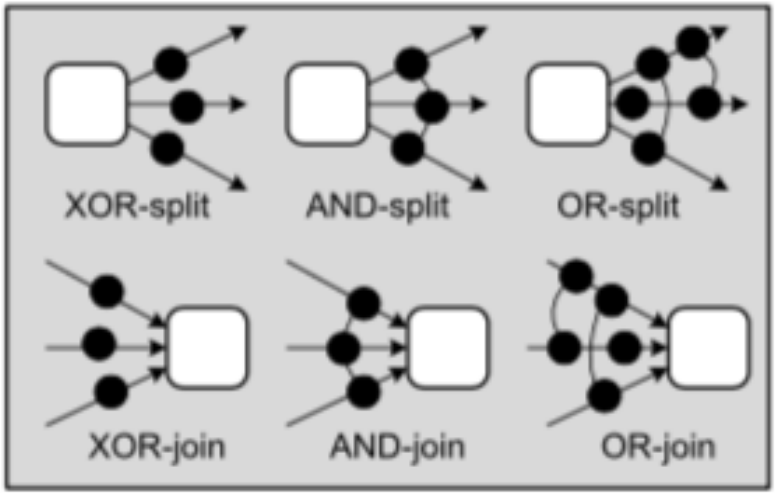
Obligations are like tokens and they need to be there in order to be consumed by input binding

Why C-nets?

- Many of the process mining algorithms that can deal with noise and other advanced concepts, they use a representation very similar to C-nets. And the primary example that we will use here is the heuristics miner.
- It fits very well with mainstream languages. They often talk about OR and AND in a more elaborate way than what is possible in Petri net model or a transition system, so there is a nice fit.
- Able to model XOR's, AND's, and very important OR's but we do so without adding any other modelling elements like silent or duplicate transitions in a Petri net.
- C-net avoid non-sound models. So the semantics are in such a way that we avoid having deadlocks in them.

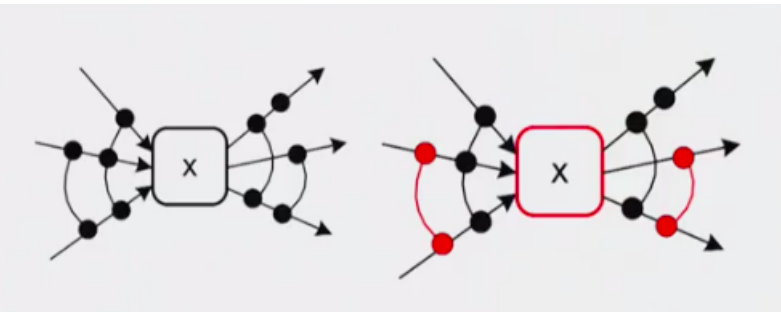
Rules of Game - One Possible Execution



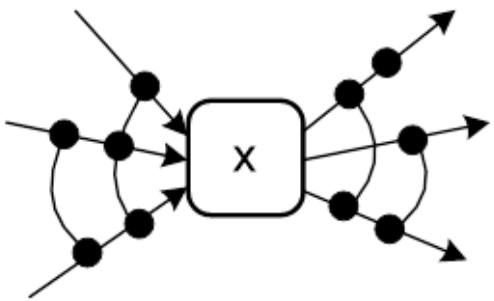


Possible Bindings - Possible Traces

Activity x has two input bindings and three output bindings and one possible execution of this activity x is highlighted here in red. That we have an input binding involving two, earlier activities and, so, there need to be two, obligations from before. Before we can do x. And x is creating two obligations into the future.

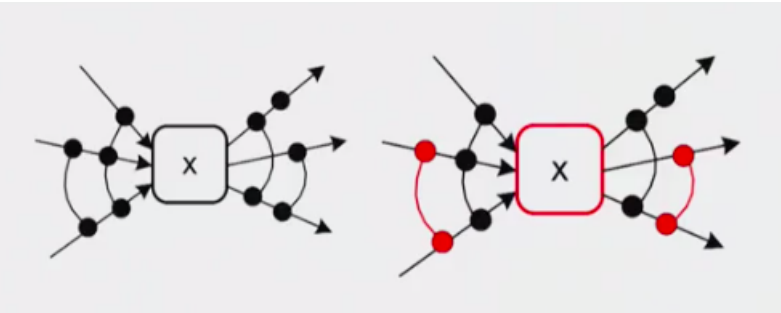


How many bindings involving this activity are possible?

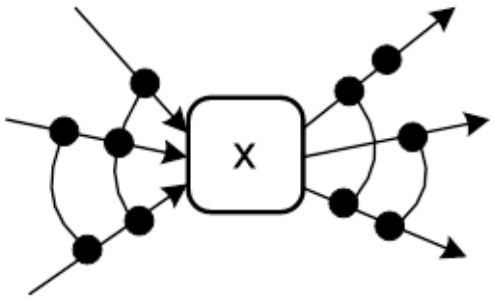


Possible Bindings - Possible Traces

Activity x has two input bindings and three output bindings and one possible execution of this activity x is highlighted here in red. That we have an input binding involving two, earlier activities and, so, there need to be two, obligations from before. Before we can do x. And x is creating two obligations into the future.

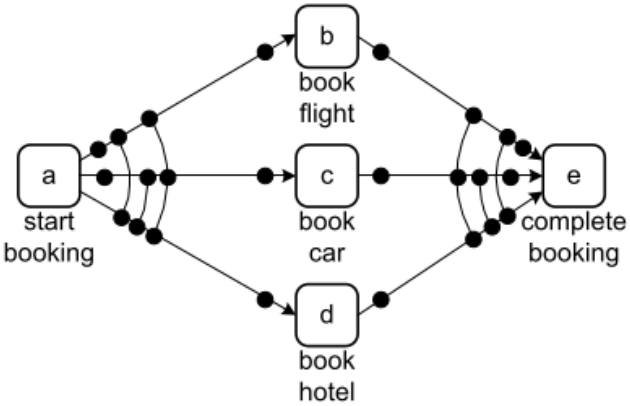


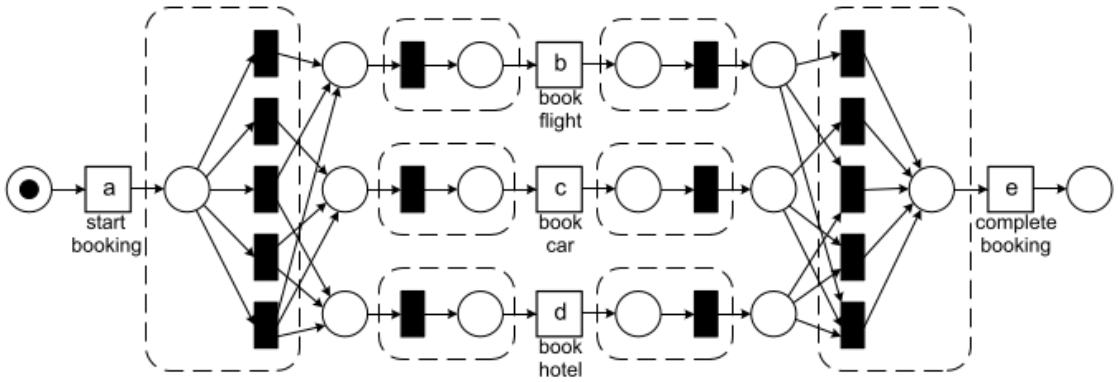
How many bindings involving this activity are possible?



6 possible bindings!

C-net to WF-Net





If we take a C-net, we can easily translate it into a Petri net that allows for the behavior that we see in the C-net

- We just take all the output bindings and convert them into silent transitions
- We take all the input bindings, and create them, we create also silent transitions for them
- We connect where the arcs are, we add places connecting the output bindings of one transition to the input binding of another transition

How do WF-Net and C-nets relate to each other?

Every valid binding sequence of the C-net corresponds to a firing sequence of the workflow net that goes from the initial marking to the final marking, we can call this a valid firing sequence

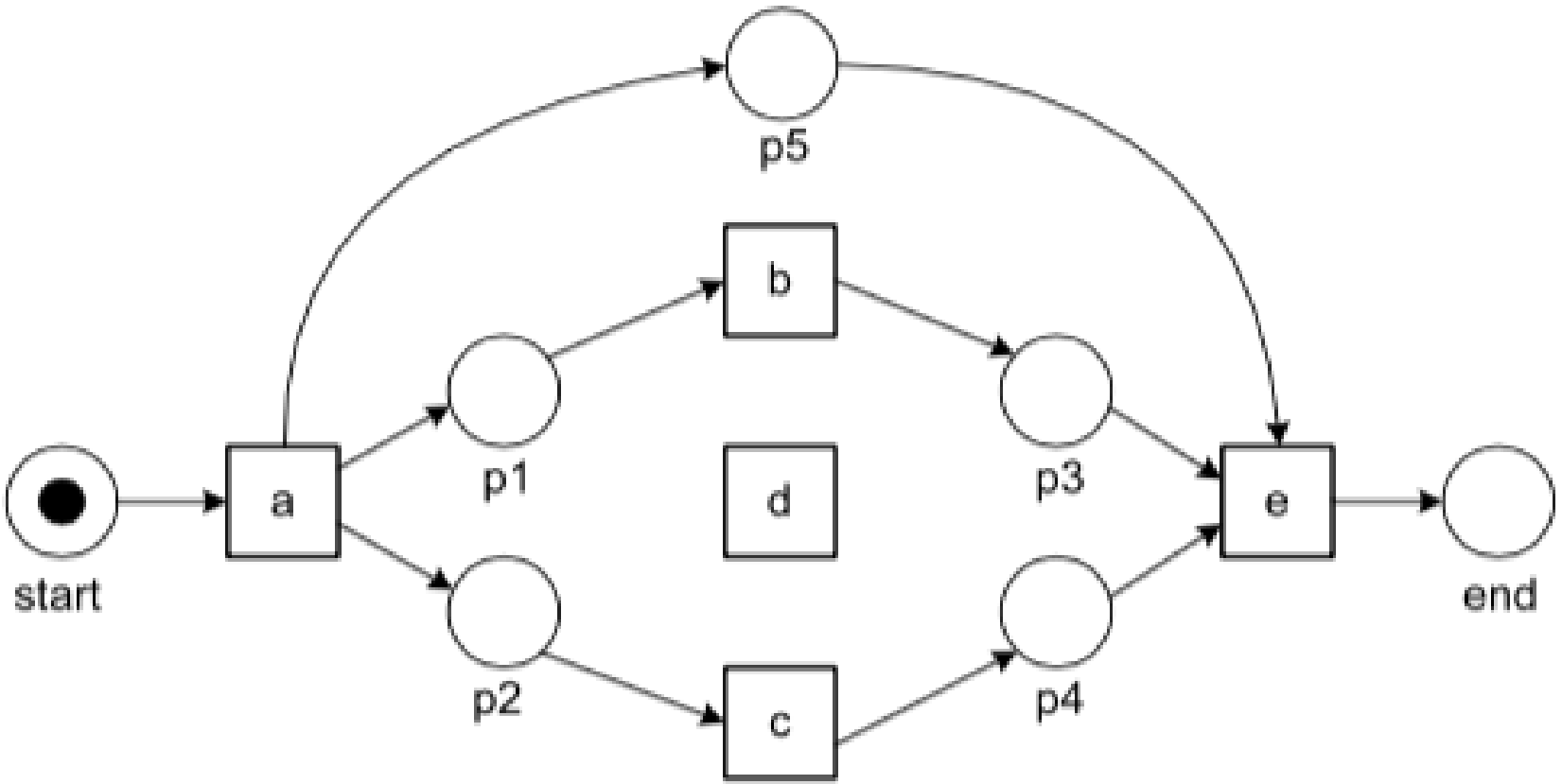
In the workflow net there may be many executions that do not correspond to a valid binding sequence, because they end up in a deadlock or unable to reach the final state.

- when we talk about C-nets, we only consider valid binding sequences
- hen we talk about workflow nets, we also are confronted with deadlocks and livelocks that need to be considered

Learning the Dependency Graph

Applying Alpha Algoritm - Is not suitable!!!

$$L = [\langle a, e\rangle^{5}, \langle a, b, c, e\rangle^{10}, \langle a, c, b, e\rangle^{10}, \langle a, b, e\rangle^{1}, \langle a, c, e\rangle^{1},$$
$$\langle a, d, e\rangle^{10}, \langle a, d, d, e\rangle^{2}, \langle a, d, d, d, e\rangle^{1}]$$



Learning the Dependency Graph

Definition 7.1 (Dependency measure) Let L be an event log[1] over $\mathscr{A}$ and $a, b \in \mathscr{A}$. $|a >_L b|$ is the number of times a is directly followed by b in L, i.e.,

$$|a >_L b| = \sum_{\sigma \in L} L(\sigma) \times \left|\left\{1 \leq i < |\sigma| \;\middle|\; \sigma(i) = a \wedge \sigma(i+1) = b\right\}\right|$$

$|a \Rightarrow_L b|$ is the value of the dependency relation between a and b:

$$|a \Rightarrow_L b| = \begin{cases} \frac{|a>_L b| - |b>_L a|}{|a>_L b| + |b>_L a| + 1} & \text{if } a \neq b \\ \frac{|a>_L a|}{|a>_L a| + 1} & \text{if } a = b \end{cases}$$

Frequency of the directly follows

$$L = [\langle a, e\rangle^{5}, \langle a, b, c, e\rangle^{10}, \langle a, c, b, e\rangle^{10}, \langle a, b, e\rangle^{1}, \langle a, c, e\rangle^{1},$$
$$\langle a, d, e\rangle^{10}, \langle a, d, d, e\rangle^{2}, \langle a, d, d, d, e\rangle^{1}]$$

$\lvert >_L \rvert$	a	b	c	d	e
a	0	11	11	13	5
b	0	0	10	0	11
c	0	10	0	0	11
d	0	0	0	4	13
e	0	0	0	0	0

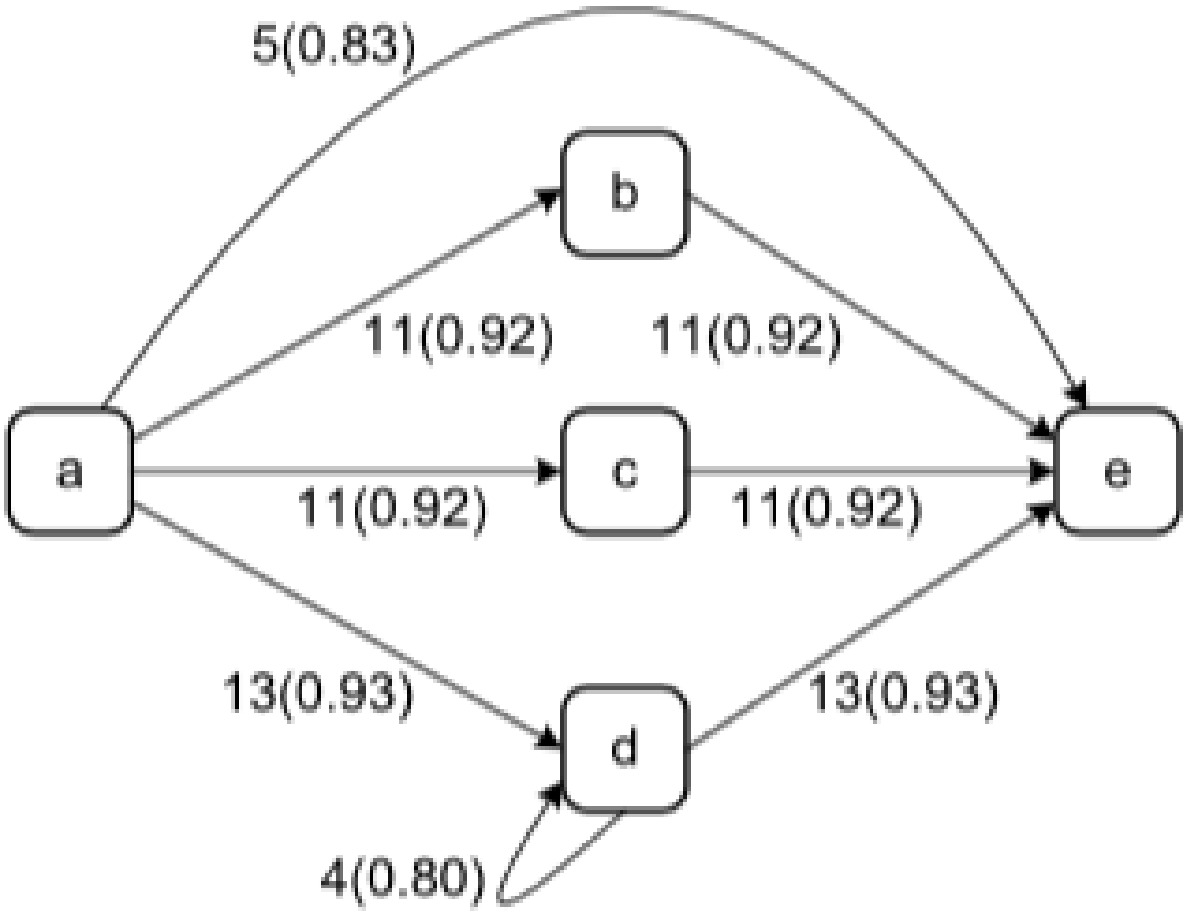
Dependency measures

$\lvert\Rightarrow_L\rvert$	a	b	c	d	e
a	$\frac{0}{0+1}=0$	$\frac{11-0}{11+0+1}=0.92$	$\frac{11-0}{11+0+1}=0.92$	$\frac{13-0}{13+0+1}=0.93$	$\frac{5-0}{5+0+1}=0.83$
b	$\frac{0-11}{0+11+1}=-0.92$	$\frac{0}{0+1}=0$	$\frac{10-10}{10+10+1}=0$	$\frac{0-0}{0+0+1}=0$	$\frac{11-0}{11+0+1}=0.92$
c	$\frac{0-11}{0+11+1}=-0.92$	$\frac{10-10}{10+10+1}=0$	$\frac{0}{0+1}=0$	$\frac{0-0}{0+0+1}=0$	$\frac{11-0}{11+0+1}=0.92$
d	$\frac{0-13}{0+13+1}=-0.93$	$\frac{0-0}{0+0+1}=0$	$\frac{0-0}{0+0+1}=0$	$\frac{4}{4+1}=0.80$	$\frac{13-0}{13+0+1}=0.93$
e	$\frac{0-5}{0+5+1}=-0.83$	$\frac{0-11}{0+11+1}=-0.92$	$\frac{0-11}{0+11+1}=-0.92$	$\frac{0-13}{0+13+1}=-0.93$	$\frac{0}{0+1}=0$

- If the number is close to one, there is a very strong causality
- If the number is negative or if the number is close to zero there is a weak casuality in that direction

We can assume threshold on the number

Dependency measures

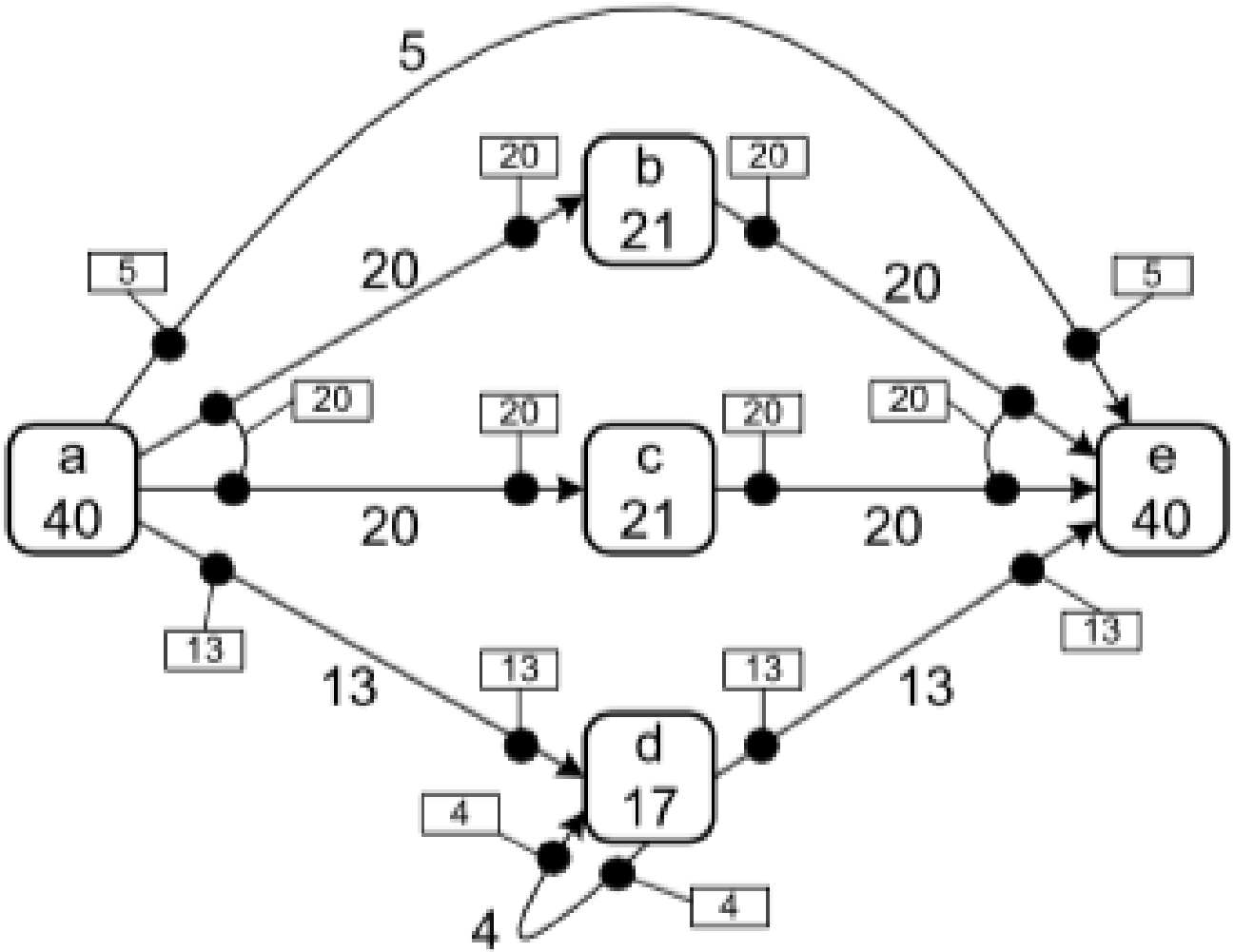


By playing with thresholds we can make the dependency graph simpler.

So, what is the algorihm?

- First we set the thresholds
- ... then we count direct successions
- ... then we compute the dependency measures using the two formulas discussed
- ... then we draw the dependency graph, including only the arcs that meet both thresholds
- ... then we get the dependency graph

Learning Splits and Joins

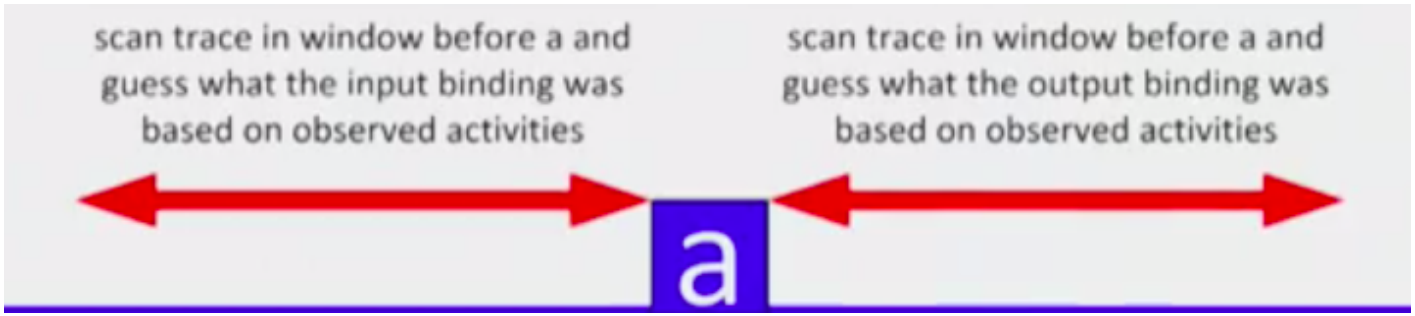


Approaches

- Heuristic using a time window before and after each activity. By counting sets of input and output activities the binding can be determined (local decision).
- Optimisation approaches based on reply. Given a set of possible input and output binding one can see whether reality can be replayed properly. The set of possible input and output binding is finite, so a best set of bindings can be determined using some goal function.

Many variations are possible!

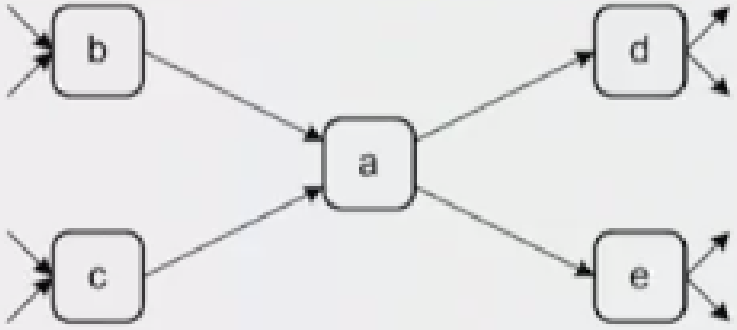
Approach 1: Based on heuristics



- Activities have possible inputs and outputs (based on dependency graph)
- Count how often they appear in a windows before (for input binding) and a window after (for output binding)

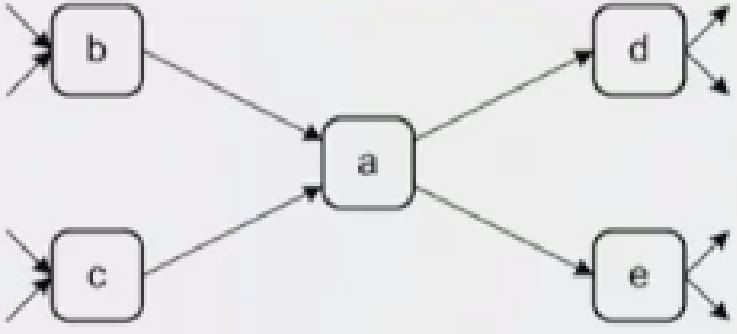
Windows size 4 - Example 1

1....klbgadhek...
2....lkgcahedl...
3....kblgaehdk...
4....klgbadehk...
5....klkcadkeh...



input bindings
- {b}: 3 times
- {c}: 2 times

1....klbgadhek...
2....lkgcahedl...
3....kblgaehdk...
4....klgbadehk...
5....klkcadkeh...



output bindings
- {d,e}: 5 times

Adding bindings and frequencies - Example 1

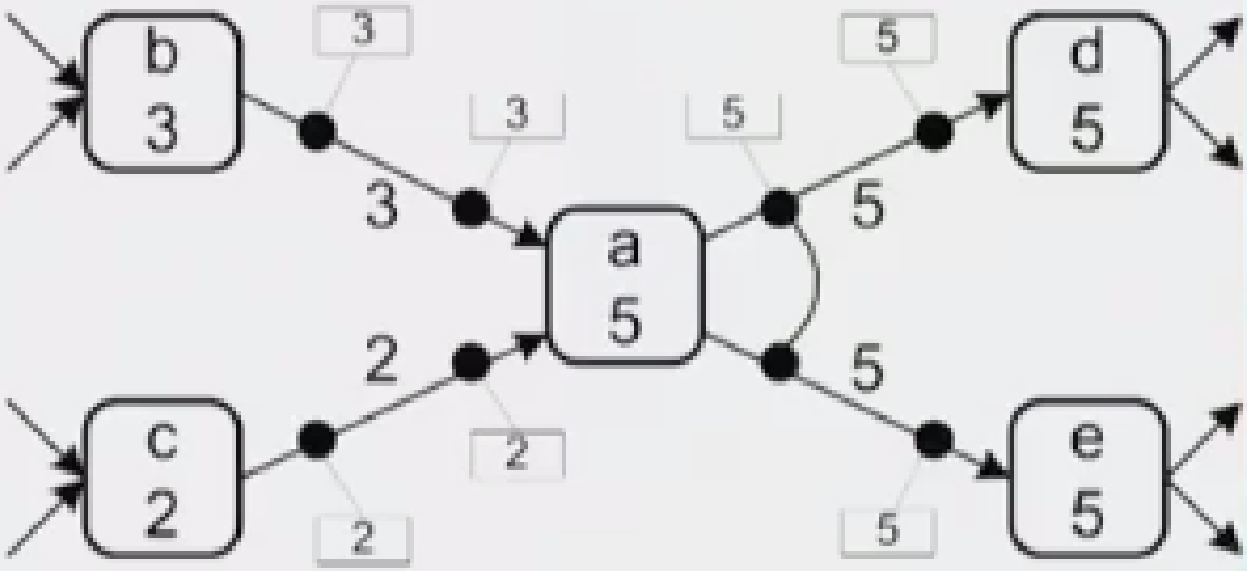
1. ...klbgadhek...
2. ...lkgcahedl...
3. ...kblgaehdk...
4. ...klgbadehk...
5. ...klkcadkeh...

input bindings

- {a}: 3 times
- {c}: 2 times

output bindings

- {d,e}: 5 times



Adding bindings and frequencies - Example 2

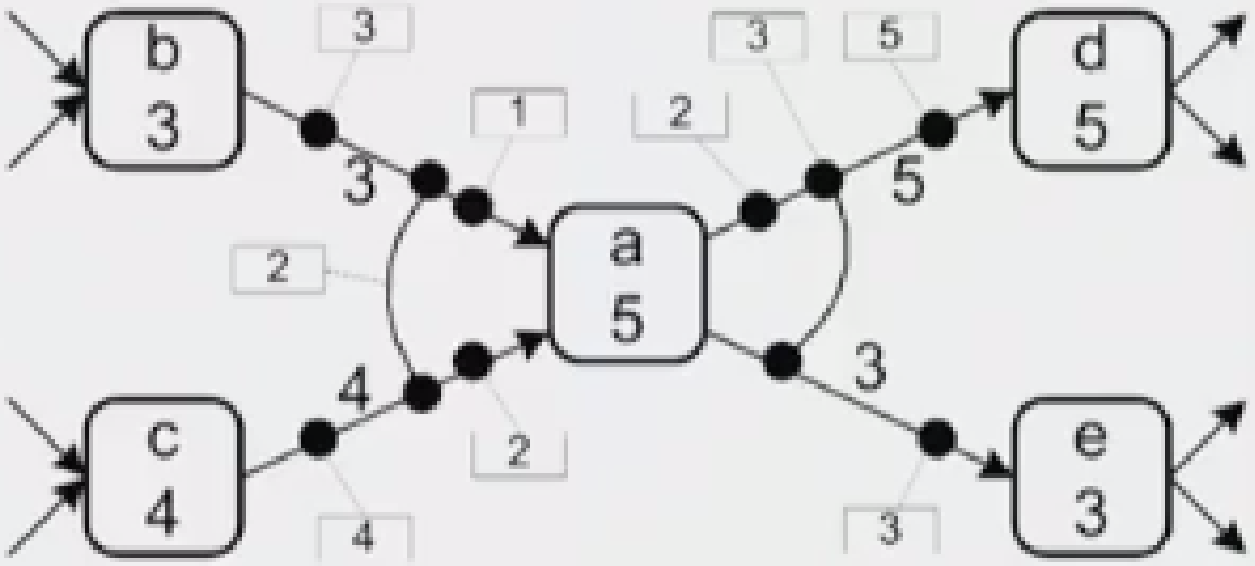
1. ...klbgadhek...
2. ...lkgcahhdl...
3. ...kbcgaehdk...
4. ...klcbadkhk...
5. ...klkcadkeh...

input bindings

- {b}: 1 time
- {c}: 2 times
- {b,c}: 2 times

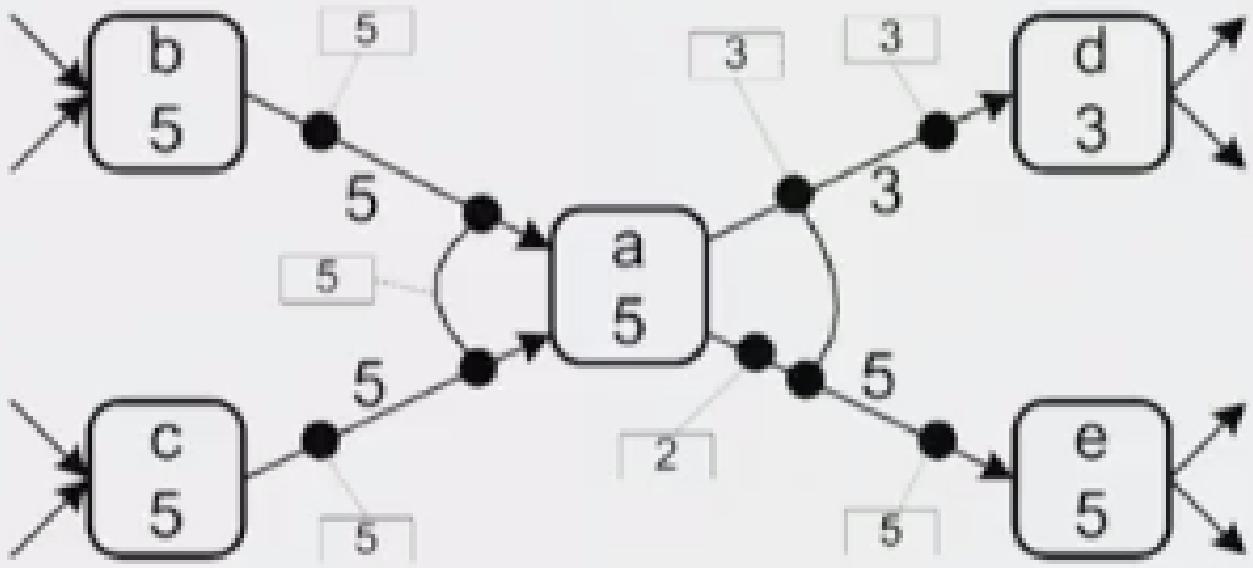
output bindings

- {d}: 2 times
- {d,e}: 3 times



Adding bindings and frequencies - Example 3

1. ...kcbgadhek...
2. ...lbgcaehdl...
3. ...bkcgaehhk...
4. ...cklbakkhe...
5. ...kbkcadkeh...



input bindings

- {b,c}: 5 times

output bindings

- {e}: 2 times
- {d,e}: 3 times

Refinements needed!

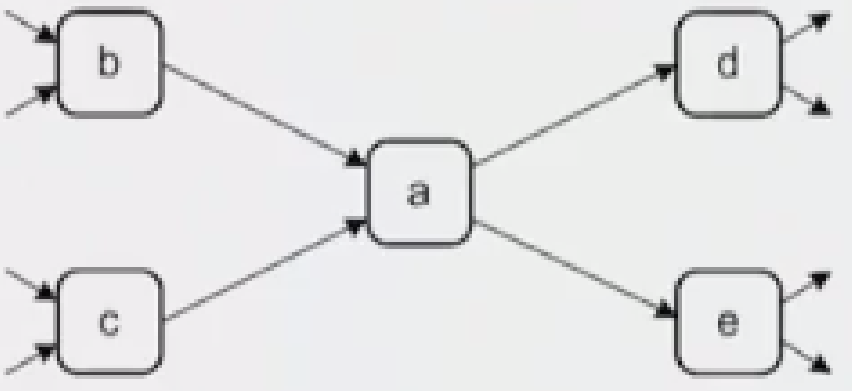
- What if there are no corresponding activities in the input or output windows?
- Noise filtering (remove infrequent bindings)
- Handling repeating activities (e.g. cut off windows size)

Approach 2: Optimisation problem

- Evaluate all possible activity bindings and take best one
- Based on the idea that ideally a trace can be replayed from the initial state to the final state
- This can be checked precisely using various replay approach
- One can use approach that simply try bindings exhaustively (evaluate and take the best one

Example: sets of input and output bindings

- Each input output arc needs to be involved in at least one binding.
- For arch activity select one of the input-output binding combination
- One can do this exhaustively and try all combinations
- Evaluation can be done using replay
- Take best one (taking into account fitness. precision, generalisation and simplicity



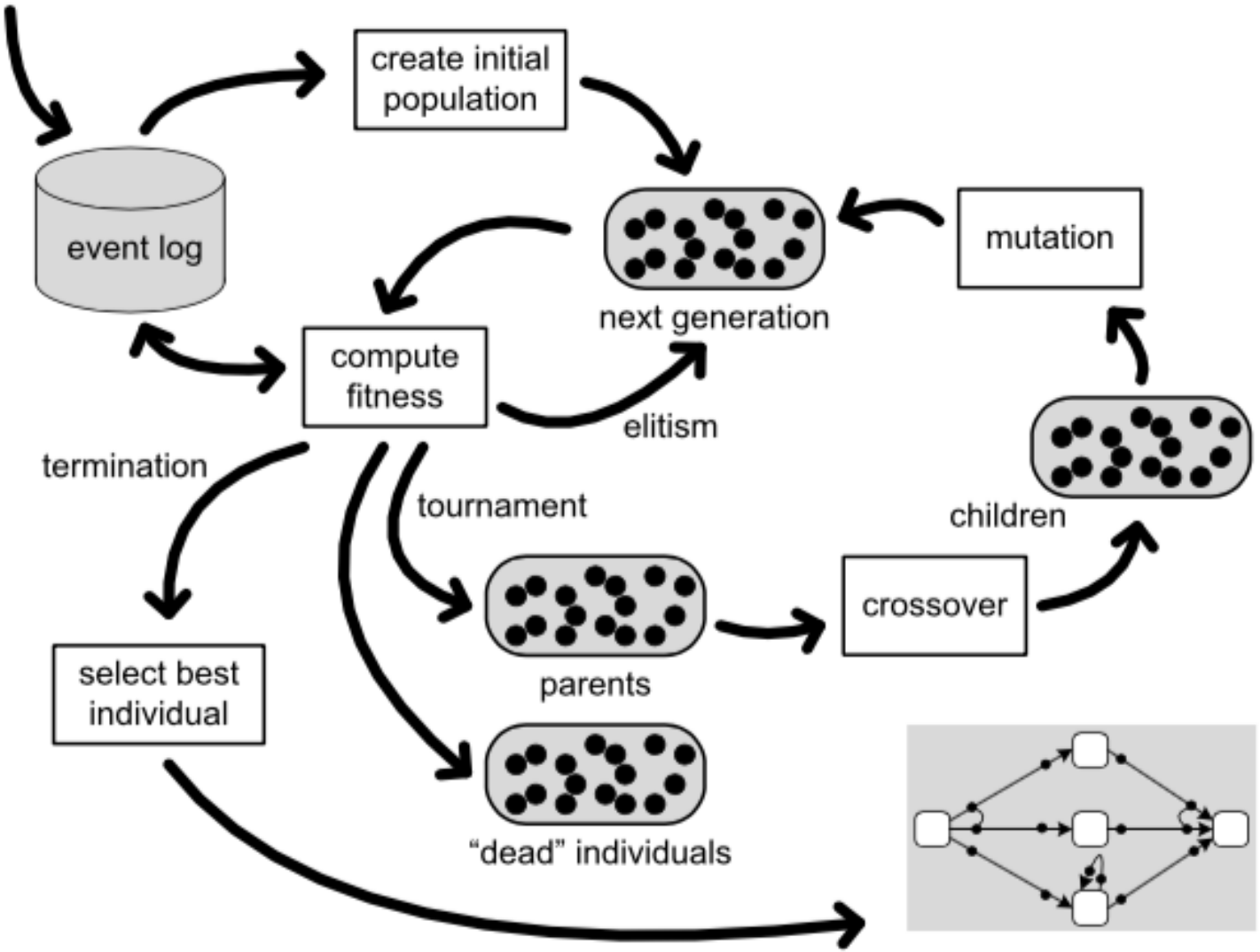
|{ {{b},{c}} , {{b},{b,c}} ,
{{c},{b,c}} , {{b},{c},{b,c}} }| x
|{ {{d},{e}} , {{d},{d,e}} ,
{{e},{d,e}} , {{d},{e},{d,e}} }|
= 4 x 4 = 16 possible a activities

IF too time consuming, ...

- Randomize
- Use a genetic algorithm

Genetic Process Mining

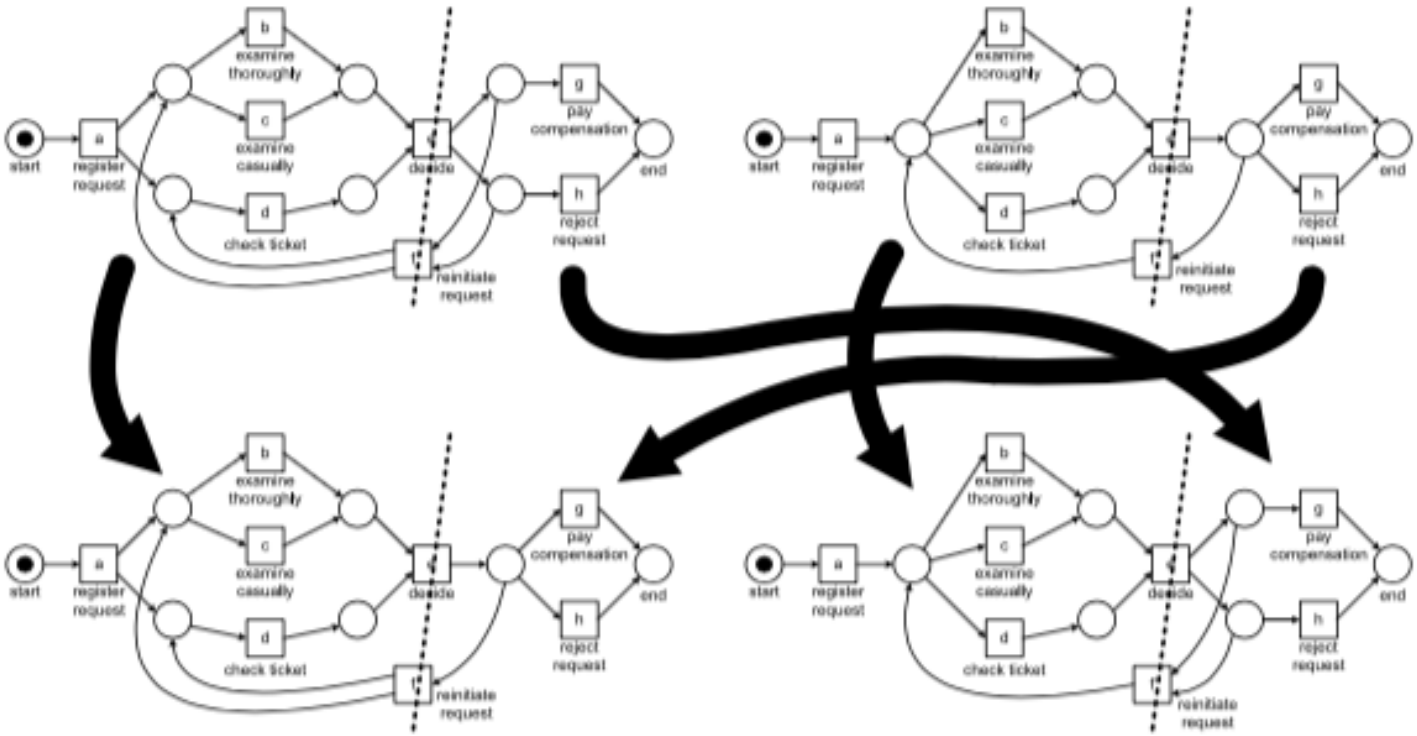
Approach used for genetic process mining



Approach used for genetic process mining

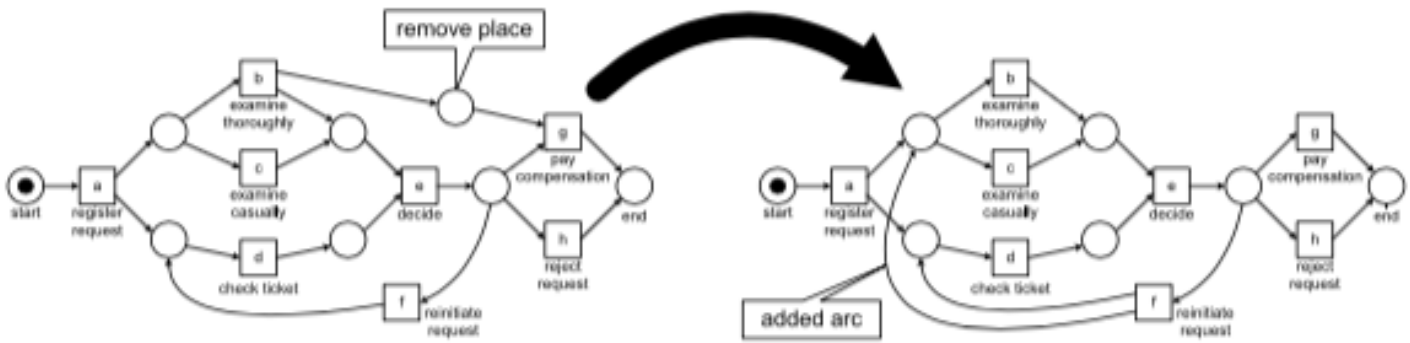
- Representation of individuals. Each individual corresponds to a process model described in a particular language, e.g., Petri nets, C-nets, BPMN, or EPCs.
- Fitness function. Here, the challenge is to define a function that balances the four competing quality criteria
- Selection strategy (tournament and elitism). The genetic algorithm needs to de- termine the fraction of individuals that go to the next round without any changes
- Crossover. The goal of crossover is to recombine existing genetic material. The basic idea is to create a new process model that uses parts of its two parent models.
- Mutation. The goal of mutation is to randomly insert new genetic material.

Crossover



Two parent models (top) and two child models resulting from a crossover. The crossover points are indicated by the dashed lines

Mutation



Mutation: a place is removed and an arc is added