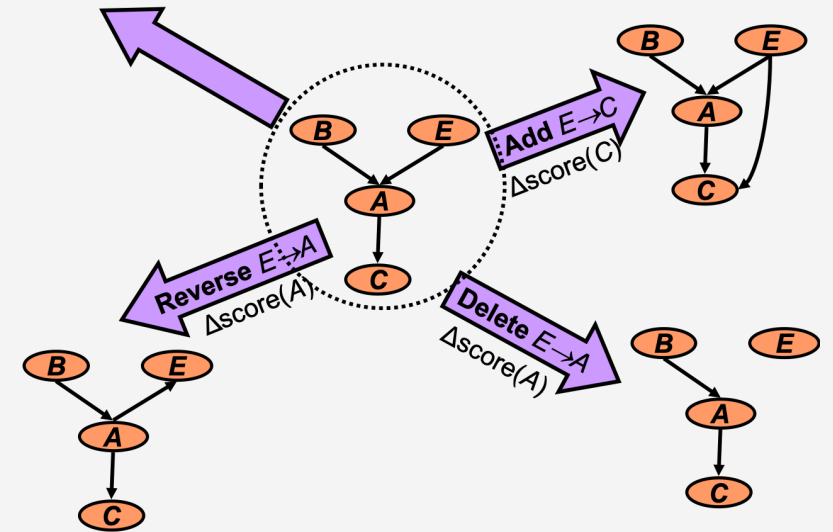


Universität
Münster

Learning

Statistical Relational Artificial Intelligence
(StaRAI)



Contents

1. Introduction

- Notions of AI, agents, StaRAI

2. Modelling Formalisms

- a. *Logic*
- b. *Probabilistic Graphical Models (PGMs)*
- c. *Probabilistic Relational Models (PRMs)*
 - Syntax, semantics, inference tasks

3. Inference

- a. *Exact Inference*
 - i. *(Weighted) Model Counting*
 - ii. *Variable Elimination*
 - iii. *Lifted Inference*

b. *Approximate Inference*

- Inference algorithms

4. Learning

- Relation learning in logic
- Parameter / structure learning for PGMs
- From PGMs to PRMs, PRM learning

5. Sequential Models and Inference

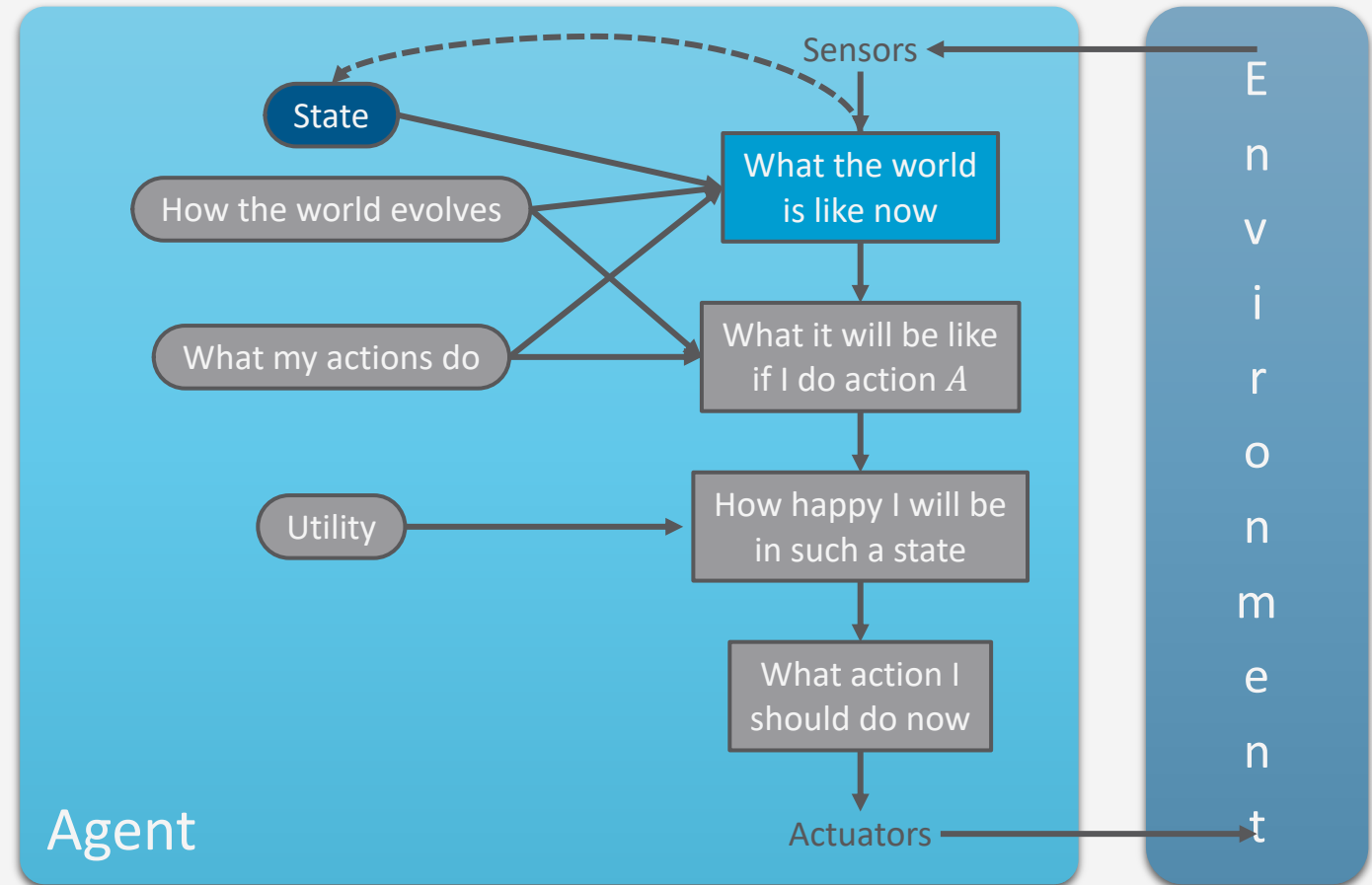
- Syntax, semantics, inference tasks, algorithms

6. Decision-theoretic Models and Inference

- Preferences, utility
- Syntax, semantics, inference task, algorithm

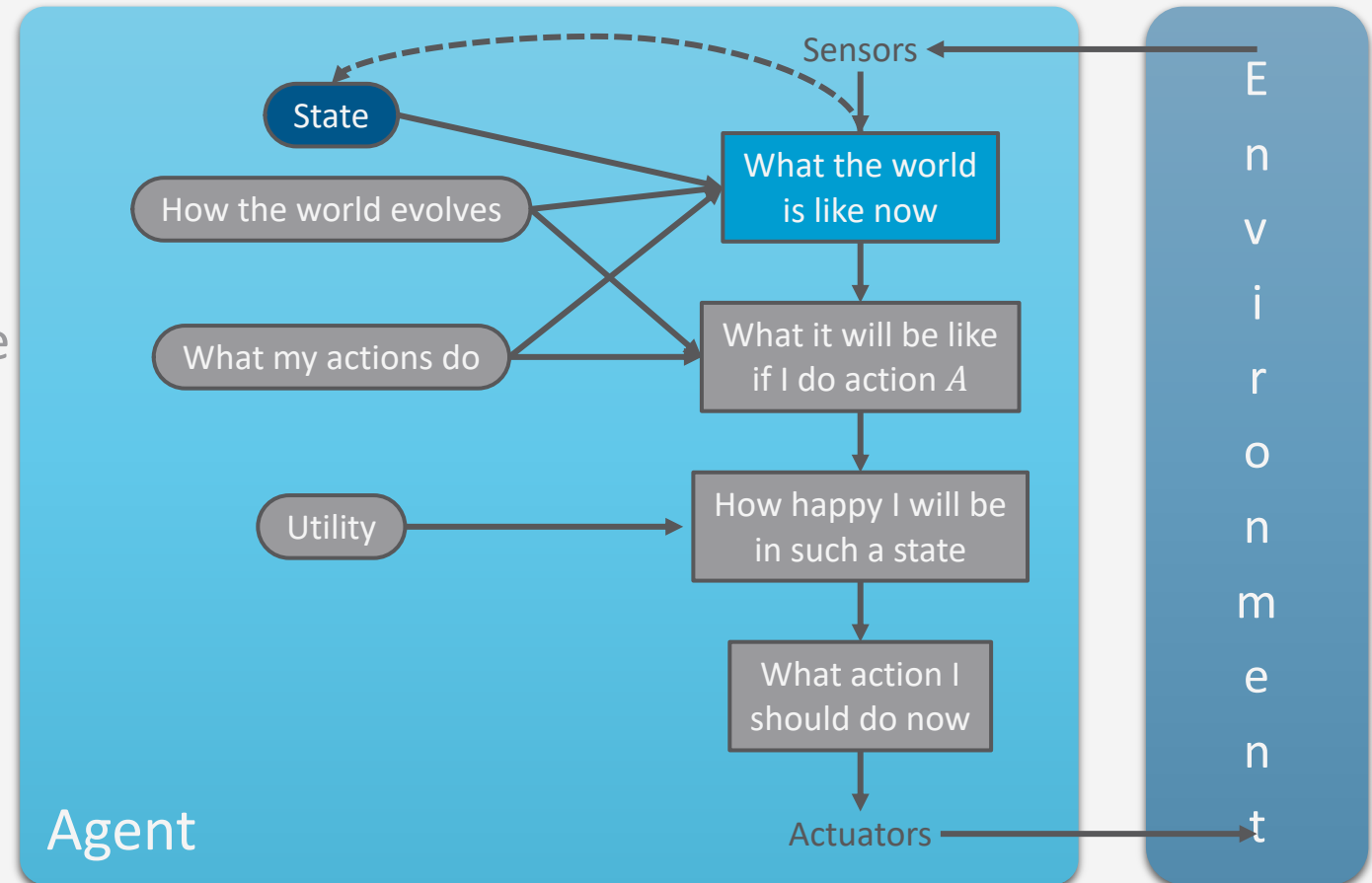
Contents in this Lecture Related to *Utility-based Agents*

- Further topics
 2. Modelling formalisms
 3. Inference (in episodic models)
 4. Learning (of episodic models)
 5. Sequential models and inference
 6. Decision-theoretic models and inference



Contents in this Lecture Related to *Utility-based Agents*

- Further topics
 3. (Episodic) PRMs
 4. Lifted inference (in episodic PRMs)
 5. Lifted learning (of episodic PRMs)
 6. Lifted sequential PRMs and inference
 7. Lifted decision making
 8. Continuous space and lifting



Generative & Discriminative Models

- Methods described are for learning models representing a full joint $P(X, Y)$
 - One can do any kind of inference (prediction, classification, any probability of random variables) one is interested in
- **Generative** models
 - Also allows to sample data, i.e., *generate* new data points
- In contrast: **Discriminative** models (such as neural nets)
 - Specifically designed and trained to maximise performance of classification: $P(Y|X)$
 - Y a classification random variable and X a vector of features
 - Generally perform better on classification than generative models when given a reasonable amount of training data
 - By focusing on modelling a conditional distribution

In both cases: Models are an abstraction/generalisation of data, which cannot represent the data with 100% accuracy. Only the data can do that.

Outline: 4. Learning

a. Relation Learning in Logic

- First-order inductive learning, first-order logical decision trees

b. Learning of PGMs

- Parameter and structure learning

c. Lifting PGMs to PRMs

- Colour passing

d. PRM Learning

- Relational dependency network learning
- Changing domains

Relation Learning in Logic

- Logic perspective on learning
 - Examples are composed of descriptions and classifications
 - Objective is to find a hypothesis that explains the classification of the examples, given their descriptions
 - $\text{Hypothesis} \wedge \text{Descriptions} \models \text{Classifications}$

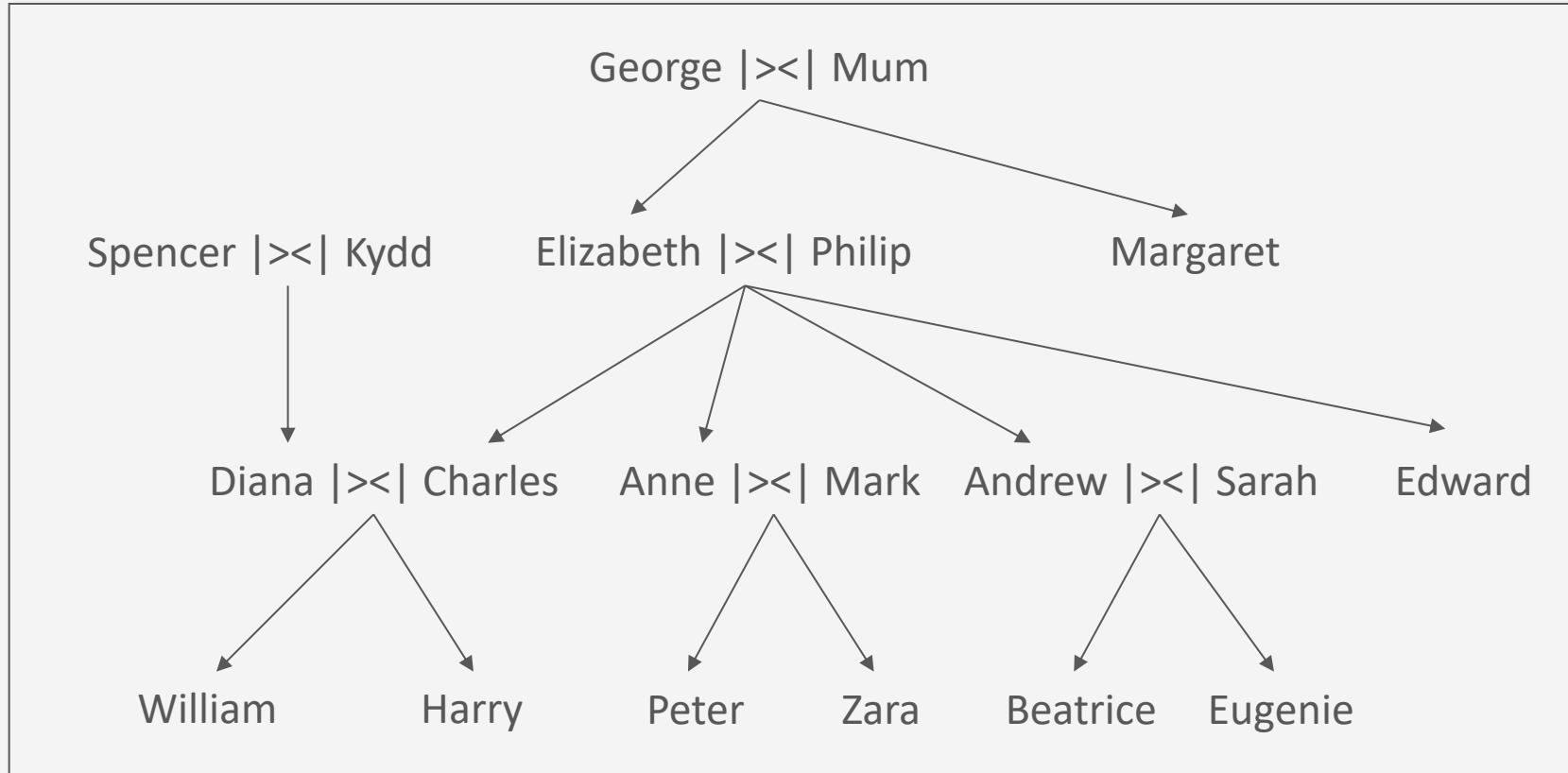
Knowledge-based Inductive Learning

- *Background knowledge* helps to explain examples
 - $\text{Background} \wedge \text{Hypothesis} \wedge \text{Descriptions} \models \text{Classifications}$
 - E.g., inferring disease D from symptoms not enough to explain prescription of medicine M
 - Rule that M is effective against D needed
- Using knowledge, effective hypothesis space reduced to include only those theories consistent with what is already known
 - Prior knowledge can be used to reduce size of hypothesis explaining the observations
 - Smaller hypotheses easier to find
- Main research field: inductive logic programming (ILP)

First-order Inductive Learning (FOIL)

- Learns function-free Horn clauses for a target concept given a set of positive and negative examples and some background knowledge
 - Form of ILP
 - Form of top-down learning
 - Start from a general rule and specialize it
- E.g., learning family relations from examples
 - Observations are an extended family tree
 - *Mother*, *Father*, and *Married* relations
 - *Male* and *Female* properties
 - Target predicates, e.g., *Grandparent*, *BrotherInLaw*, *Ancestor*

A Not-so-recent Example



Example: *Grandparent*

Background $\wedge$ Hypothesis $\wedge$ Descriptions $\models$ Classifications

- Descriptions

include facts like

Father(Philip, Charles), Mother(Mum, Margaret)

Married(Diana, Charles), Male(Philip), Female(Beatrice)

- Sentences in **Classifications** depend on the target concept being learned

- In the example: 12 positive, 388 negative

Grandparent(Mum, Charles), $\neg$ Grandparent(Mum, Harry)

- Goal: find a set of sentences for **Hypothesis** such that the entailment constraint is satisfied

- E.g., without background knowledge, hypothesis is:

$$\begin{aligned} \text{Grandparent}(x, y) \Leftrightarrow & [\exists z \text{ Mother}(x, z) \wedge \text{Mother}(z, y)] \\ & \vee [\exists z \text{ Mother}(x, z) \wedge \text{Father}(z, y)] \\ & \vee [\exists z \text{ Father}(x, z) \wedge \text{Mother}(z, y)] \\ & \vee [\exists z \text{ Father}(x, z) \wedge \text{Father}(z, y)] \end{aligned}$$

Background Knowledge

$$\text{Background} \wedge \text{Hypothesis} \wedge \text{Descriptions} \models \text{Classifications}$$

- A little bit of **background** knowledge helps a lot

- E.g.,

- Background knowledge contains

$$\text{Parent}(x, y) \Leftrightarrow [\text{Mother}(x, y) \vee \text{Father}(x, y)]$$

- Grandparent is now reduced to

$$\text{Grandparent}(x, y) \Leftrightarrow [\exists z \text{Parent}(x, z) \wedge \text{Parent}(z, y)]$$

- Constructive induction algorithm
 - Create new predicates to facilitate the expression of explanatory hypotheses
 - E.g.,
 - Introduce a predicate *Parent* to simplify the definitions of the target predicates

FOIL: *Grandparent* Example

- Split positive and negative examples
- Construct set of Horn clauses with *Grandfather*(x, y) as head with positive examples as instances of *Grandfather* relationship
 - Start with a clause with an empty body
 - All examples classified as positive, so specialise to rule out negative examples
 1. Incorrectly classifies all positive examples
 2. Incorrect on larger part of negative examples
 3. Prefer the third clause; further specialise
- Positive:
<George, Anne>, *<Philip, Peter>*, *<Spencer, Harry>*, ...
- Negative:
<George, Elizabeth>, *<Harry, Zara>*, *<Charles, Philip>*, ...
- Start:
_____ $\Rightarrow$ *Grandfather*(x, y)
- 3 potential additions:
 1. *Father*(x, y) $\Rightarrow$ *Grandfather*(x, y)
 2. *Parent*(x, z) $\Rightarrow$ *Grandfather*(x, y)
 3. *Father*(x, z) $\Rightarrow$ *Grandfather*(x, y)
- Further specialisation:
Father(x, z) $\wedge$ *Parent*(z, y) $\Rightarrow$ *Grandfather*(x, y)

FOIL: Algorithm

function FOIL(*examples*, *target*) **returns** a set of Horn clauses

inputs: *examples*, set of examples

target, a literal for the goal predicate

local variables: *clauses*, set of clauses, initially empty

while *examples* contains positive examples **do**

clause $\leftarrow$ New-Clause(*examples*, *target*)

remove examples covered by *clause* from *examples*

add *clause* to *clauses*

return *clauses*

FOIL

- Function New-Clause: generate clause covering all positive examples while excluding as many negative examples as possible

FOIL: New Clause

function New-Clause(*examples, target*) **returns** a Horn clause

local variables:

clause, a clause with target as head and an empty body

l, a literal to be added to the clause

extended, a set of examples with values for new variables

extended $\leftarrow$ *examples*

while *extended* contains negative examples **do**

l $\leftarrow$ Choose-Literal(New-Literals(*clause*), *extended*)

append *l* to the body of clause

extended $\leftarrow$ set of examples created by applying

Extend-Example to each example in *extended* for *l*

return *clause*

FOIL: new clause

FOIL: New Literals

- New—Literals: generates a set of new literals to possibly be added to the body of a clause
 - Input: *clause*, a clause
 - Output: *literals*, a set of literals
- E.g.,
 $Father(x, z) \Rightarrow Grandfather(x, y)$
 - Using predicates
 - Valid: *Mother(z, u)*, *Married(z, z)*, *Grandfather(v, x)*, *Parent(z, y)*
 - Invalid: *Married(u, v)*
 - Inequality: $z \neq x$
 - Arithmetic comparisons: $x > y$ (not meaningful here)
- Approach: Add to *literals*
 - Using predicates:
 - Negated or unnegated
 - Use any existing predicate (including the goal)
 - By allowing target predicate here, learn recursive definitions, but keep from infinite recursion
 - Arguments must be variables
 - Each literal must include at least one variable from an earlier literal or from the head of the clause
 - Tests for (in)equality of variables already occurring in the rule
 - Test on empty lists
 - Arithmetic comparisons
 - Also on threshold values

FOIL: Choose Literal

- Choose—Literal: heuristic function that chooses a literal out of a set of literals
 - Input:
 - *literals*, a set of literals to choose from
 - *extended*, a set of positive and negative examples
 - Possibly any other input required for making a decision
 - Output: *literal*, the chosen literal
 - Approach: Base decision on a criterion such as [information gain](#)
 - How much better can one distinguish the positive and the negative examples given the current clause R_0 compared to an extended version R_1 with the literal added to the body of the clause
$$Gain(R_0, R_1) = t \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$
 - p_i, n_i denote the number of positive, negative examples covered by R_i (classified as positive by R_i)
 - t denotes the number of positive examples covered by both
 - See also: [Information theory, entropy, and information gain for decision trees](#)

FOIL: Extend Example

```
function Extend–Example(example, literal) returns a set of examples  
  if example satisfies literal then  
    return the set of examples created by extending example with  
      each possible constant value for each new variable in literal  
  else  
    return the empty set
```

FOIL: extend example

FOIL: Optimisations

- The way New—Literal changes the clauses leads to a very large branching factor
 - Improve performance by using type information
 - E.g., *Parent*(x, n) where x is a person and n is a number
 - **Ockham's razor** to eliminate hypotheses
 - If a clause becomes longer than the total length of the positive examples that the clause explains, the clause is not a valid hypothesis
- Rules/FOL formulas have to satisfy all positive examples while excluding all negative examples
 - Otherwise inconsistent
 - Combine with probabilities or weights to reflect inconsistency and uncertainty

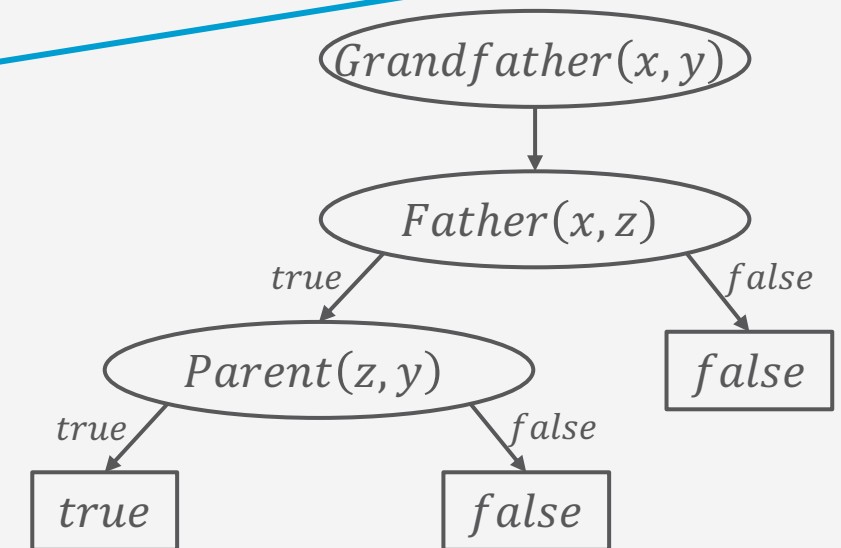
Decision Tree Representation

- Represent result as decision tree
 - Target (head) as root, followed by decision nodes (body)
 - (Conjunctions of) literals in inner nodes
 - Left child: path from root to inner node evaluates to *true*
 - Right child: path from root to inner node evaluates to *false*
 - Different nodes can share variables under the restriction that a variable introduced in a node must not occur in right branch of that node
 - Leaves: indicate if path is a model
 - Rework to contain class labels
→ first-order logical decision tree
- E.g.,

$$Father(x, z) \wedge Parent(z, y) \Rightarrow Grandfather(x, y)$$

Follows from semantics of tree:

- Variable X introduced in a node is existentially quantified within the conjunction of that node
- Right subtree only relevant if conjunction fails (“there is no such X ”), in which case further reference to X is meaningless



First-order Logical Decision Trees

- Instead of learning a logic program, learn a **first-order logical decision tree, FOLDT**
 - Logical representation of a *relational decision tree*
 - Input: examples, background knowledge, target concept (classes)
 - *true / false* in the FOIL setting
 - Output: FOLDT
 - Defined as on previous slide with leaves containing class names
 - Idea:
 - Choose (a conjunction of) literals at each inner node such that the examples are split up in groups that are as homogeneous as possible with respect to classes occurring (very idea of decision trees)
- Called *learning from interpretations*
 - Also what ProbLog does

FOLDT: Example

- Idea of what is to learn:
 - Check a machine with parts X
 - If machine contains worn parts that cannot be replaced by engineer, send back to manufacturer
 - If all worn parts can be replaced, then fix it
 - No worn parts, ok
- Learning progress: to the right below
- Resulting logic program:

```

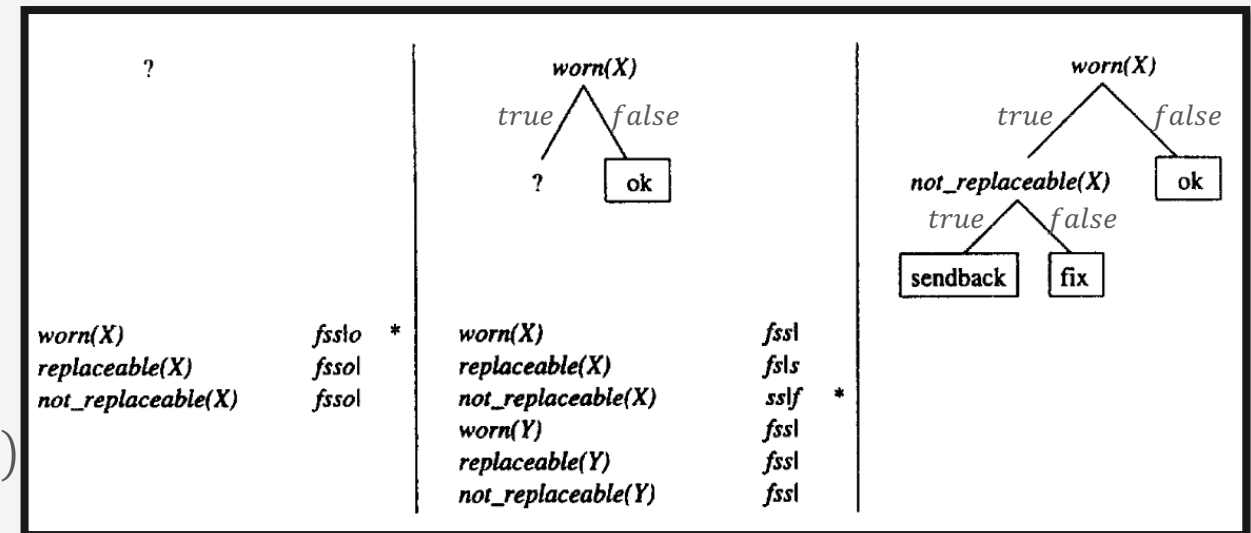
class(ok)           ←  $\forall X : \neg worn(X)$ 
class(sendback) ←  $\exists X : worn(X) \wedge not\_replaceable(X)$ 
class(fix)          ←  $\exists X : worn(X) \wedge$ 
                      $\forall Y : (\neg worn(Y) \vee \neg not\_replaceable(X))$ 

```

Input examples, background knowledge

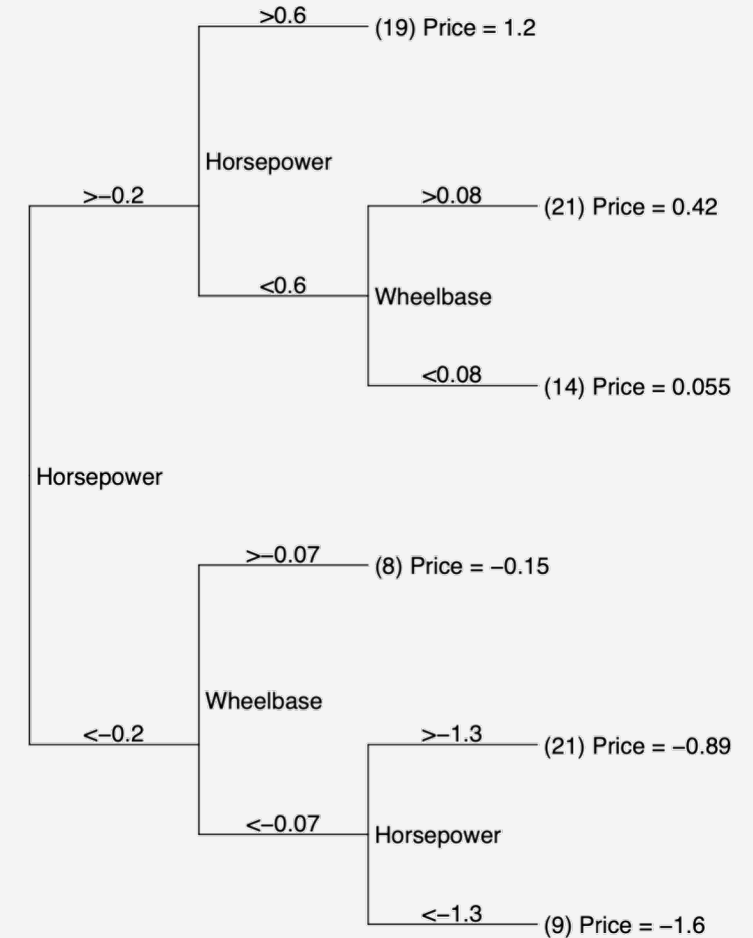
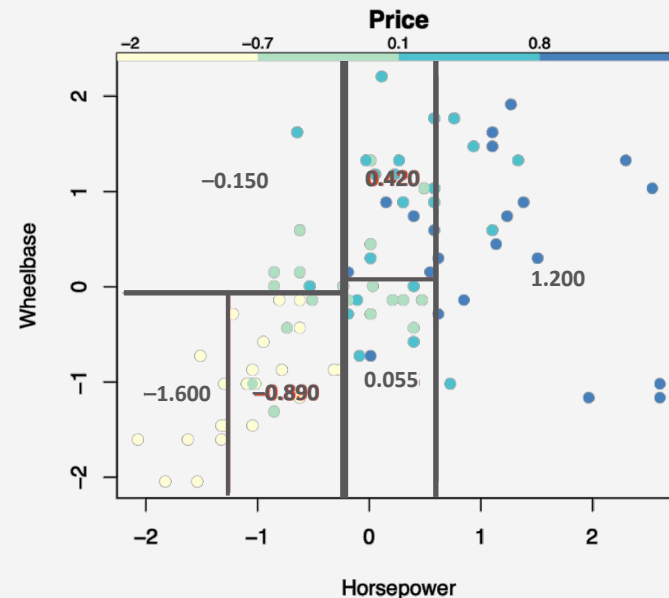
Example 1	Example 2	Example 3	Example 4
<i>class(fix)</i>	<i>class(sendback)</i>	<i>class(sendback)</i>	<i>class(ok)</i>
worn(gear)	worn(engine)	worn(wheel)	
worn(chain)	worn(chain)		

Background knowledge
replaceable(gear)
replaceable(chain)
not_replaceable(engine)
not_replaceable(wheel)



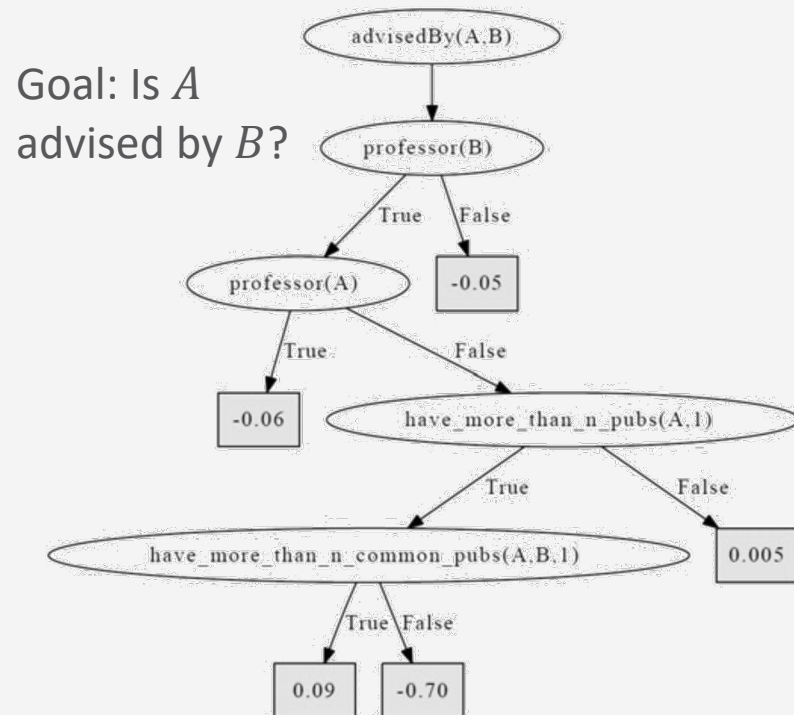
Regression Trees

- Regression trees = decision trees with continuous values (regression values) in leaves
 - Could base decision on variance
 - Depends on application how regression values are calculated
 - E.g., predict price of cars
 - Regression values = average

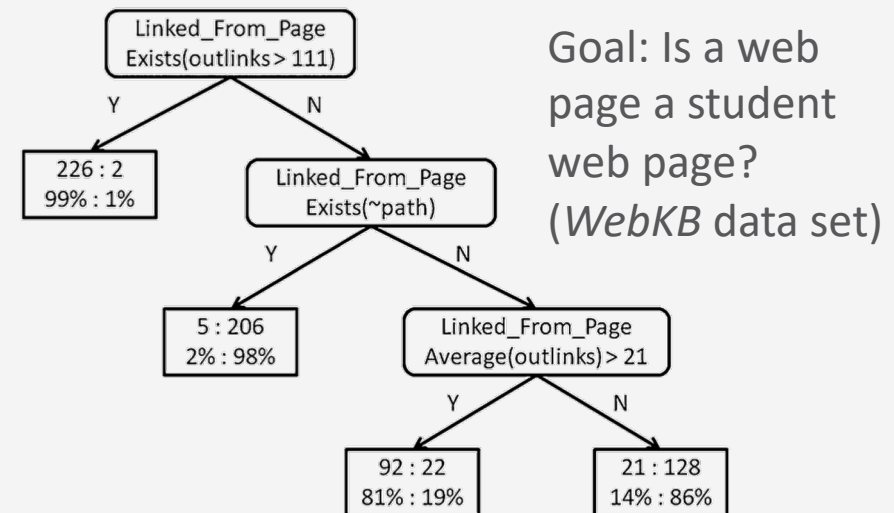


Relational Regression & Probability Trees

- Relational regression tree (RRT)
 - FOLDT with continuous values in leaves



- Relational probability tree (RPT)
 - $\approx$ FOLDT with *probability distributions* in leaves



There are some differences what they allow inner nodes to be

- Not important to grasp the general idea

Interim Summary

- FOIL
 - Learn relations given examples, background knowledge and a target concept
- First-order logical decision trees
 - Learn relations in tree form given examples, background knowledge and a target concept

Outline: 4. Learning

a. Relation Learning in Logic

- First-order inductive learning, first-order logical decision trees

b. Learning of PGMs

- Parameter and structure learning

c. Lifting PGMs to PRMs

- Colour passing

d. PRM Learning

- Relational dependency network learning
- Changing domains

Learning PGMs

Learning Setting

- So far: query answering task given a model
→ form of predicting data
- New task: Given a set of samples (data), learn a model
- Question: Which model is the “right” one?
 - Model which makes data most likely:
Reproduce given data with high probability
(by *sampling*)
 - Sub-questions: Are all random variables (features) known? Is data available for all?

Dimensions

- **Parameter estimation**
 - Find the best parameters (probabilities in CPTs / potentials in factors)
 - One set of parameter values called **hypothesis**
 - Easier if data available for all variables
- **Structure search** (on top of parameter est.)
 - Find the best structure representation
 - Correctly represent (in)dependencies
 - Which random variables are connected?
Should we introduce new variables?

Parameter Estimation: Approaches

- Problem: Parameter search space continuous $\rightarrow$ Infinite number of hypotheses
- **Full Bayesian Learning**: Consider all possible values for the parameters if possible
 - Means that we work with the set of all possible models $\rightarrow$ query answering needs to consider all models and combine answers of all (weighted by likelihood of model)

$$P(X | \mathbf{d}) \propto \sum_{\boldsymbol{\theta}} P(X | \mathbf{d}, \boldsymbol{\theta}) P(\boldsymbol{\theta} | \mathbf{d}) = \sum_{\boldsymbol{\theta}} P(X | \boldsymbol{\theta}) P(\mathbf{d} | \boldsymbol{\theta}) P(\boldsymbol{\theta}) = \sum_{\boldsymbol{\theta}} P(X | \boldsymbol{\theta}) P(\boldsymbol{\theta}) \prod_{j=1}^N P(d_j | \boldsymbol{\theta})$$

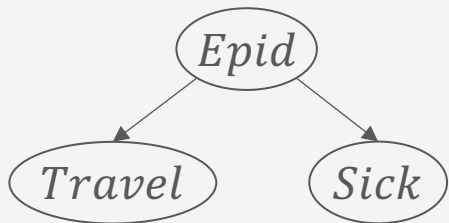
- **MAP Learning**: choose the MAP hypothesis given data
 - $\boldsymbol{\theta}_{MAP} = \arg \max_{\boldsymbol{\theta}} P(\mathbf{d} | \boldsymbol{\theta}) P(\boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} P(\boldsymbol{\theta}) \prod_{j=1}^N P(d_j | \boldsymbol{\theta})$
- **ML Learning**: choose ML hypothesis
 - $\boldsymbol{\theta}_{ML} = \arg \max_{\boldsymbol{\theta}} P(\mathbf{d} | \boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} \prod_{j=1}^N P(d_j | \boldsymbol{\theta})$
- MAP + ML: only one set of parameter values $\boldsymbol{\theta}$, i.e., one model: Query answering as before

independent and identically
distributed (iid) data points

Learning in BNs: Data Science with Complete Data

- We begin with the simplest learning problem: Learning parameters in BNs with
 - Known structure
 - Data contains observations for all variables
 - All variables observable, no missing data

Model



Data

<i>Travel</i>	<i>Epid</i>	<i>Sick</i>
false	false	false
false	false	true
true	false	false
false	true	true
⋮	⋮	⋮

→ Probability distributions to learn

$P(\text{Epid}), P(\text{Travel} \mid \text{Epid}), P(\text{Sick} \mid \text{Epid})$

<i>Epid</i>	<i>P</i>
false	$1 - \theta_0$
true	θ_0

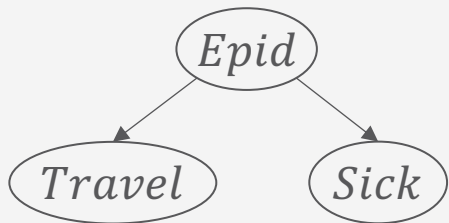
<i>Epid</i>	<i>Travel</i>	<i>P</i>
false	false	$1 - \theta_1$
false	true	θ_1
true	false	$1 - \theta_2$
true	true	θ_2

<i>Epid</i>	<i>Sick</i>	<i>P</i>
false	false	$1 - \theta_3$
false	true	θ_3
true	false	$1 - \theta_4$
true	true	θ_4

Maximum-Likelihood Parameter Estimation

- Simplest learning problem
 - Assumed that BN structure known
 - Data contains observations for all variables
- Goal: Estimate BN parameters θ
 - Entries in CPTs $P(R|\text{Pa}(R))$

Model



Data

<i>Travel</i>	<i>Epid</i>	<i>Sick</i>
<i>false</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>false</i>	<i>true</i>
<i>true</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>true</i>
$\vdots$	$\vdots$	$\vdots$

→ Probability distributions to learn

$P(\text{Epid}), P(\text{Travel} \mid \text{Epid}), P(\text{Sick} \mid \text{Epid})$

<i>Epid</i>	<i>P</i>
<i>false</i>	$1 - \theta_0$
<i>true</i>	θ_0

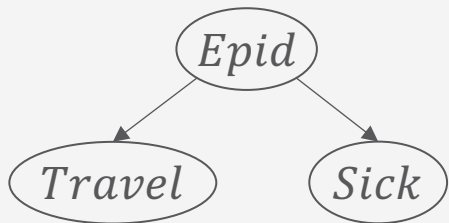
<i>Epid</i>	<i>Travel</i>	<i>P</i>
<i>false</i>	<i>false</i>	$1 - \theta_1$
<i>false</i>	<i>true</i>	θ_1
<i>true</i>	<i>false</i>	$1 - \theta_2$
<i>true</i>	<i>true</i>	θ_2

<i>Epid</i>	<i>Sick</i>	<i>P</i>
<i>false</i>	<i>false</i>	$1 - \theta_3$
<i>false</i>	<i>true</i>	θ_3
<i>true</i>	<i>false</i>	$1 - \theta_4$
<i>true</i>	<i>true</i>	θ_4

General ML procedure

- **Maximum Likelihood Estimation** (MLE) principle: Choose θ such that data most probable
- Procedure
 1. Express the likelihood of the data as a function of the parameters θ to be learned
 2. Take the derivative of the log likelihood with respect to each parameter in θ
 3. Set derivatives equal to 0 and solve for each parameter in θ

Model



Data

Travel	Epid	Sick
false	false	false
false	false	true
true	false	false
false	true	true
⋮	⋮	⋮

→ Probability distributions to learn

$P(\text{Epid}), P(\text{Travel} \mid \text{Epid}), P(\text{Sick} \mid \text{Epid})$

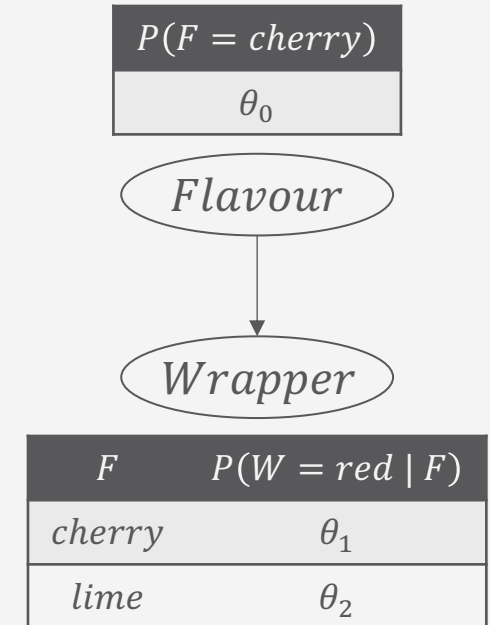
Epid	P
false	$1 - \theta_0$
true	θ_0

Epid	Travel	P
false	false	$1 - \theta_1$
false	true	θ_1
true	false	$1 - \theta_2$
true	true	θ_2

Epid	Sick	P
false	false	$1 - \theta_3$
false	true	θ_3
true	false	$1 - \theta_4$
true	true	θ_4

An Even Smaller Example

- Candy factory:
 - Manufacturer chooses colour of candy wrapper probabilistically for each candy based on flavour, following an unknown distribution
 - If *Flavour* = *cherry*, chooses *Wrapper* = *red* with probability θ_1
 - If *Flavour* = *lime*, chooses *Wrapper* = *red* with probability θ_2
 - BN for this problem includes three parameters to learn
 - $\theta_0, \theta_1, \theta_2$



An Even Smaller Example

- Probability for, e.g., $W = \text{yellow}, F = \text{cherry}$

$$\begin{aligned} & P(W = \text{yellow}, F = \text{cherry} | h_{\theta_0 \theta_1 \theta_2}) \\ &= P(W = \text{yellow} | F = \text{cherry}, h_{\theta_0 \theta_1 \theta_2}) P(F = \text{cherry} | h_{\theta_0 \theta_1 \theta_2}) \\ &= (1 - \theta_1) \theta_0 \end{aligned}$$

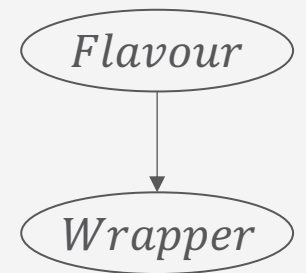
- We unwrap N candies

- Each trial gives us a data point on wrapper and flavour
- There are c cherry candies and l lime candies ($c + l = N$)
- c_r are cherry with red wrapper, c_y are cherry with yellow wrapper ($c_r + c_y = c$)
- l_r are lime with red wrapper, l_y are lime with yellow wrapper ($l_r + l_y = l$)

- Probability of data:

$$P(\mathbf{d} | h_{\theta_0 \theta_1 \theta_2}) = \prod_{j=1}^N P(d_j | h_{\theta_0 \theta_1 \theta_2}) = \theta_0^c (1 - \theta_0)^l \theta_1^{c_r} (1 - \theta_1)^{c_y} \theta_2^{l_r} (1 - \theta_2)^{l_y}$$

$P(F = \text{cherry})$
θ_0



F	$P(W = \text{red} F)$
cherry	θ_1
lime	θ_2

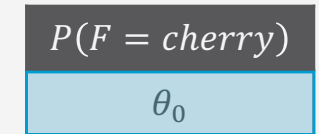
An Even Smaller Example

- ML hypothesis: values for $\theta_0\theta_1\theta_2$ such that $P(\mathbf{d}|h_{\theta_0\theta_1\theta_2}) = \prod_{j=1}^N P(d_j|h_{\theta_0\theta_1\theta_2})$ maximal
- Possible to get by maximising **log-likelihood**:

$$L(\mathbf{d}|h_{\theta_0\theta_1\theta_2}) = \log P(\mathbf{d}|h_{\theta_0\theta_1\theta_2}) = \sum_{j=1}^N \log P(d_j|h_{\theta_0\theta_1\theta_2})$$

- Advantage: Product turns into a sum
 - Usually easier to maximise
 - For the example, taking the logarithm means:

$$\begin{aligned} & L(\mathbf{d} | h_{\theta_0\theta_1\theta_2}) \\ &= \log (\theta_0^c (1 - \theta_0)^l \theta_1^{c_r} (1 - \theta_1)^{c_y} \theta_2^{l_r} (1 - \theta_2)^{l_y}) \\ &= \log \theta_0^c + \log(1 - \theta_0)^l + \log \theta_1^{c_r} + \log(1 - \theta_1)^{c_y} + \log \theta_2^{l_r} + \log(1 - \theta_2)^{l_y} \\ &= c \log \theta_0 + l \log(1 - \theta_0) + c_r \log \theta_1 + c_y \log(1 - \theta_1) + l_r \log \theta_2 + l_y \log(1 - \theta_2) \end{aligned}$$



F	P(W = red F)
cherry	θ_1
lime	θ_2

An Even Smaller Example

- Take derivatives w.r.t. θ and determine maximal values by setting derivatives equal to 0
- In the example:

- $$L(d | h_{\theta_0 \theta_1 \theta_2}) = c \log \theta_0 + l \log(1 - \theta_0) + c_r \log \theta_1 + c_y \log(1 - \theta_1) + l_r \log \theta_2 + l_y \log(1 - \theta_2)$$

- Derivative w.r.t. θ_0

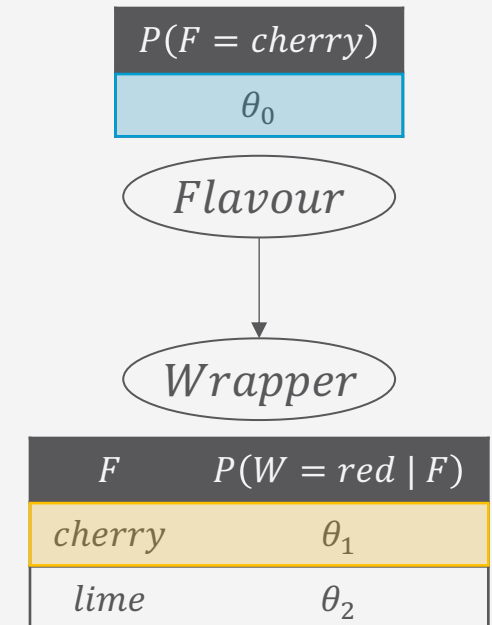
$$\frac{\partial L}{\partial \theta_0} = \frac{c}{\theta_0} - \frac{l}{1 - \theta_0} = 0 \quad \Rightarrow \quad \theta_0 = \frac{c}{c + l}$$

- Derivative w.r.t. θ_1

$$\frac{\partial L}{\partial \theta_1} = \frac{c_r}{\theta_1} - \frac{c_y}{1 - \theta_1} = 0 \quad \Rightarrow \quad \theta_1 = \frac{c_r}{c_r + c_y}$$

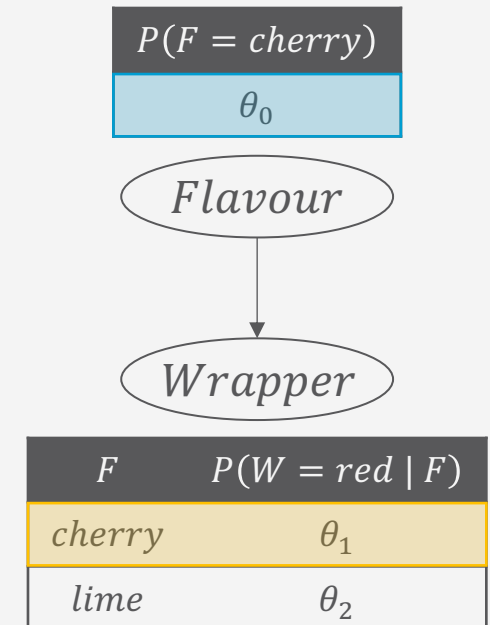
- Derivative w.r.t. θ_2

$$\frac{\partial L}{\partial \theta_2} = \frac{l_r}{\theta_2} - \frac{l_y}{1 - \theta_2} = 0 \quad \Rightarrow \quad \theta_2 = \frac{l_r}{l_r + l_y}$$



Maximum-Likelihood Parameter Estimation

- Estimating parameters by computing relative frequencies
- Process applicable to each fully observable BN
- With complete data and MLE:
 - Parameter learning decomposes into separate learning problem for each parameter (in each CPT) because of taking the logarithm
 - Each parameter $\theta = P(v|\text{pa}(V))$ in the CPT of a node V given by frequency of the considered value v given the relevant parent values $\text{pa}(V)$

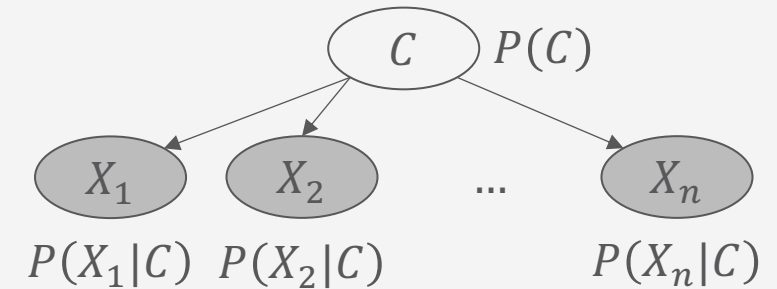


Maximum-Likelihood Parameter Estimation: General Procedure

- Lot of mathematical effort to detect the obvious
- Sketches a general procedure for estimating parameters ML-style:
 - Tasks: Maximise $P(\mathbf{d}|\boldsymbol{\theta})$, i.e., $h_{ML} = \arg \max_{\boldsymbol{\theta}} P(\mathbf{d}|\boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} \prod_{j=1}^N P(d_j|\boldsymbol{\theta})$
 - Procedure
 1. Describe probabilities of data as a function of the parameters to be learning ($P(\mathbf{d} | \boldsymbol{\theta})$)
 2. Differentiate the log-likelihood ($\log P(\mathbf{d} | \boldsymbol{\theta})$) w.r.t. each parameter
 3. Solve derivatives set equal to zero for each parameter
 - Afterwards, determine the necessary relative frequencies (i.e., counting)
- Note: Sampling can be interpreted as ML parameter estimations
 - Learning of the queried distribution using samples by determining relative frequencies

Application: Naïve Bayes Classifier

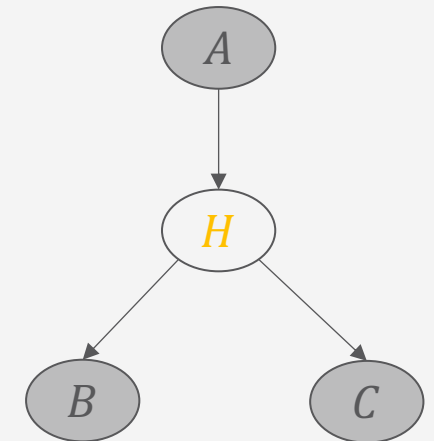
- Classification:
 - Set of features $X_1, \dots, X_n$
 - Determine class label $C = c$ based on feature values $x_1, \dots, x_n$
 - Assumption: conditional independence of features given C
- Advantage: Relatively few parameters (entries in the CPTs) to learn
 - $2n + 1$ parameters for Boolean variables (general: $r(r - 1)n + r - 1$)
 - $r = |\text{Val}(C)|$, $r - 1$ entries in $P(C)$, $r(r - 1)$ entries in $P(X_i|C)$ (for each c , learn $(r - 1)$ entries)
- Form of supervised learning; i.e., labelled data available for learning):
 - Set of feature values with corresponding class label $\mathbf{D} = \left\{ (x_1^j, \dots, x_n^j, c^j) \right\}_{j=1}^N$
 - Count relative frequencies of each c in $\mathbf{D}$ for $P(C)$ and of each x_i, c in $\mathbf{D}$ for $P(X_i | C)$



Parameter Learning with Hidden (Latent) Variables

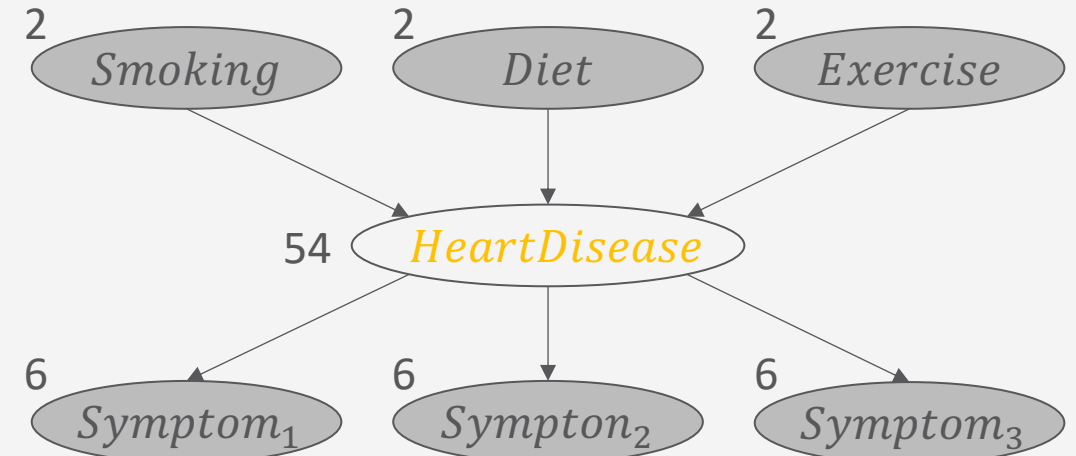
- So far we have assumed that we can collect data on all variables in the network
- Next difficult problem
 - Hidden variables in model
 - No observations available
 - Approach using relative frequencies not directly applicable as we cannot count observations for these variables
 - Example: missing all counts involving H

A	B	C
<i>true</i>	<i>false</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>true</i>	<i>false</i>
$\vdots$		



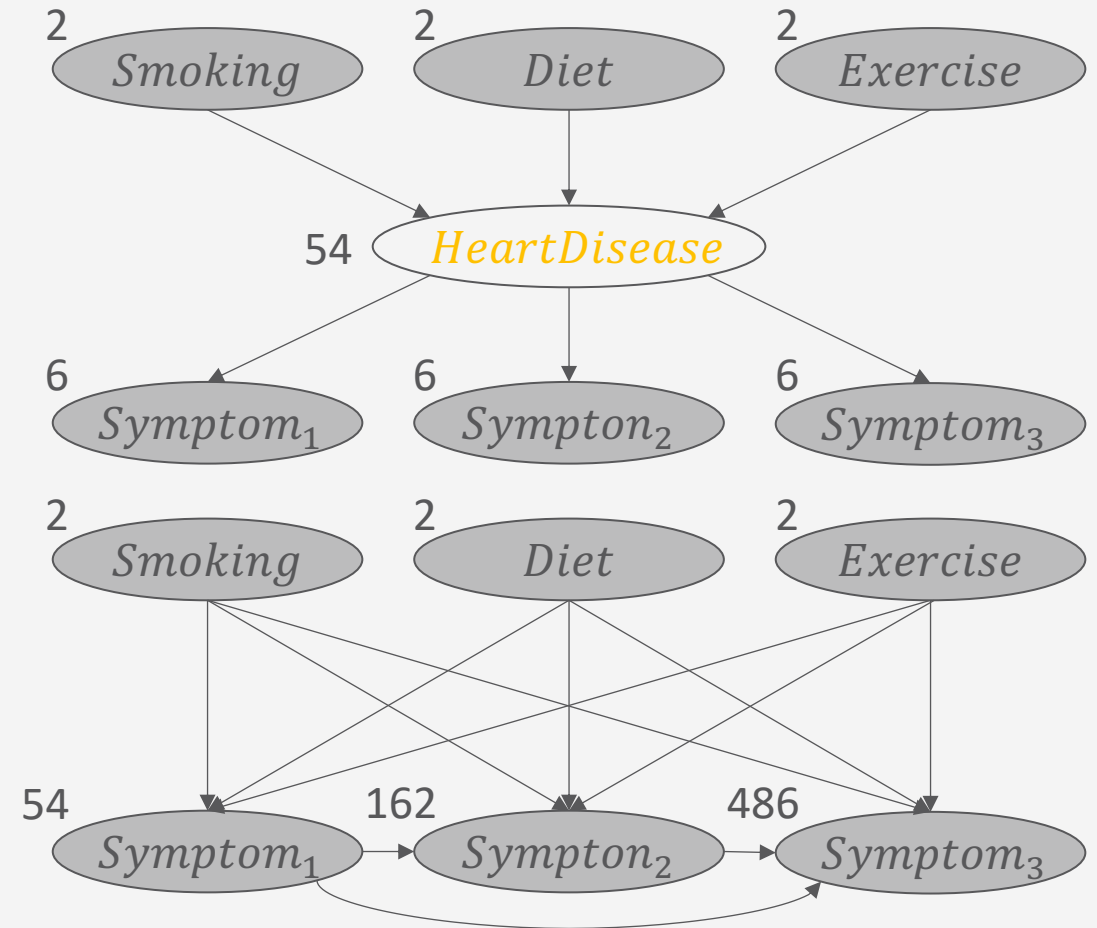
Quick Fix

- Get rid of or avoid hidden variables
- May work in simple networks
- Example
 - Each variable has 3 values (*low, moderate, high*)
 - Numbers next to the nodes represent how many parameters need to be specified for the CPTs of that node
 - 78 parameters overall



Quick Fix Does Not Help!

- Without hidden node, edges need to directly represent indirect dependencies
- In the example:
 - Symptoms no longer conditionally independent given their parents
 - Many more edges $\rightarrow$ many more parameters to learn: 708 overall
 - Need much more data to learn these parameters properly



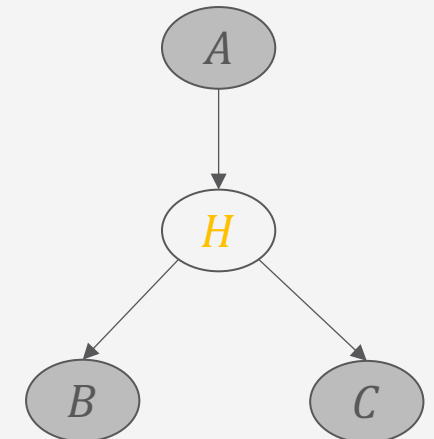
Expectation-Maximisation (EM)

- If we keep the hidden variables and want to learn parameters from data, we have a form of *unsupervised learning*
 - Data do not include information on the true nature of each data point
- **Expectation-Maximisation (EM)**
 - General algorithm for learning model parameters from incomplete data
 - Here, how it works on learning parameters for BNs with discrete variables

EM: General Idea

- If we had data for all the variables in the network, we could learn the parameters by using ML methods
 - Compute relative frequencies as discussed before
- If we had the parameters in the network, we could estimate the posterior probability of any event, including the hidden variables
 - $P(H|A, B, C)$
 - Could estimate parameters involving H

A	B	C
<i>true</i>	<i>false</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>true</i>	<i>false</i>
⋮		



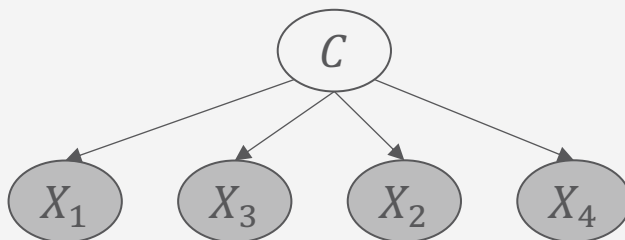
EM: General Idea

- Algorithm starts from “invented” (e.g., randomly generated) information to solve the learning problem
 - Invented parameters (virtually noted down in the CPTs)
- It then refines this initial guess by cycling through two basic steps
 - **Expectation (E):**
Update the data with predictions generated via the current model (expected counts)
 - **Maximisation (M):**
Given the updated data, update the model parameters using MLE
 - Same step as when learning parameters for fully observable networks

EM: Naïve Bayes Models as Application

- Consider the following data
 - N samples with Boolean attributes $X_1, \dots, X_4$
 - ... which we want to label with possible values of a class C , $\text{Val}(C) = \{1, 2, 3\}$
- Naïve Bayes classifier with hidden variable C

Model



→ probability distributions to learn

$P(C)$?
 $P(X_1 | C)$?
 $P(X_2 | C)$?
 $P(X_3 | C)$?
 $P(X_4 | C)$?

Data

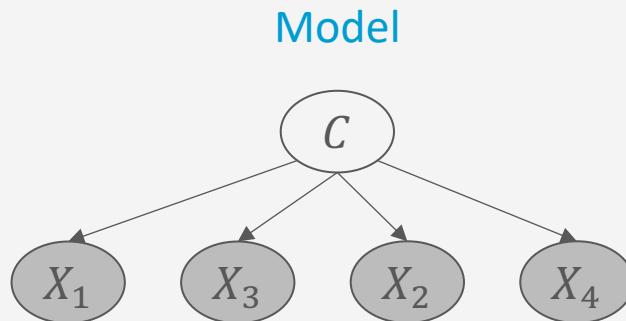
X_1	X_2	X_3	X_4
true	false	true	true
false	true	true	false
true	false	true	true
⋮			



X_1	X_2	X_3	X_4	Count
false	false	false	false	19
false	false	false	true	7
false	false	true	false	13
⋮				

EM: Initialisation

- Algorithm starts from “invented” (e.g., randomly generated) information to solve the learning problem
 - Invented parameters (virtually noted down in the CPTs)



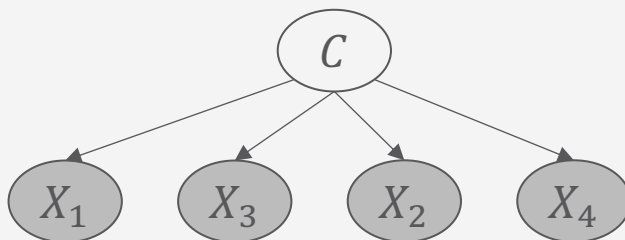
→ probability distributions to learn

$P(C)$	?	
$P(X_1 C)$	?	Define random parameters
$P(X_2 C)$	?	
$P(X_3 C)$	?	
$P(X_4 C)$	?	

EM: Expectation Step (Augment Data: Compute Expected Counts)

- What would we need to learn parameters using MLE?
 - $P(C = k) = \frac{\#(\text{datapoints with } C=k)}{\#(\text{all datapoints})}$ for each $k = 1, 2, 3$
 - $P(X_i = x \mid C = k) = \frac{\#(\text{datapoints with } X_i=x \text{ and } C=k)}{\#(\text{datapoints with } C=k)}$ for each $x \in \text{ran}(X_i), i = 1, \dots, 4, k = 1, 2, 3$
 - But, so far, we only have: $N = \#(\text{all datapoints})$
 → Approximate all other necessary counts with *expected counts*, derived from the model with „invented“ parameters

Model



Data

X_1	X_2	X_3	X_4	Zähler
false	false	false	false	19
false	false	false	true	7
false	false	true	false	13
⋮				

→ probability distributions to learn

$$\begin{aligned}
 P(C) &? \\
 P(X_1 \mid C) &? \\
 P(X_2 \mid C) &? \\
 P(X_3 \mid C) &? \\
 P(X_4 \mid C) &?
 \end{aligned}$$

EM: Expectation Step (Augment Data: Compute Expected Counts)

- Expected count $\hat{N}(C = k)$ is the sum, over all N samples, of the probability that each sample is in category k

$$\hat{N}(C = k) = \sum_{j=1}^N P(C = k \mid \underbrace{x_1^j, x_2^j, x_3^j, x_4^j}_{\text{Feature values of } j\text{'th sample}})$$

- Estimation then is:

$$P(C = k) = \frac{\text{exp\#(Datenpunkte mit } C = k)}{\text{\#(alle Datenpunkte)}} = \frac{\hat{N}(C = k)}{N}$$

- How do we get the necessary probabilities for $\hat{N}$?

Use invented parameters

- In Naïve Bayes model easy to compute:

$$P(C = k \mid x_1^j, x_2^j, x_3^j, x_4^j) = \frac{P(C = k, x_1^j, x_2^j, x_3^j, x_4^j)}{P(x_1^j, x_2^j, x_3^j, x_4^j)} = \frac{\overbrace{P(x_1^j \mid C = k) \dots P(x_4^j \mid C = k) P(C = k)}^{\text{Use invented parameters}}}{\underbrace{P(x_1^j, x_2^j, x_3^j, x_4^j)}_{\text{Also calculate using invented parameters}}}$$

EM: Expectation Step (Augment Data: Compute Expected Counts)

- Similarly, we obtain expected counts of samples with feature values $X_i = x$ and category k

$$\hat{N}(X_i = x, C = k) = \sum_{j=1}^N P(C = k | x_i^j = x, x_{i'}^j, x_{i''}^j, x_{i'''}^j)$$

- $i, i', i'', i''' \in \{1, 2, 3, 4\}, i \neq i' \neq i'' \neq i'''$

Also computed using invented parameters

- Example:

$$\hat{N}(X_1 = \text{true}, C = 1) = \sum_{e_j = (x_1, x_2, x_3, x_4), x_1 = \text{true}} P(C = 1 | x_1^j = \text{true}, x_2^j, x_3^j, x_4^j)$$

- Estimate $P(X_i | C)$

$$P(X_i = x | C = k) = \frac{\text{exp\#(datapoints with } X_i = x, C = k)}{\text{exp\#(datapoints with } C = k)} = \frac{\hat{N}(X_i = x, C = k)}{\hat{N}(C = k)}$$

- for each $x \in \text{ran}(X_i), i = 1, \dots, 4, k = 1, 2, 3$

EM: General Idea

- Algorithm starts from “invented” (e.g., randomly generated) information to solve the learning problem
 - Invented parameters (virtually noted down in the CPTs)
- It then refines this initial guess by cycling through two basic steps
 - **Expectation (E):**
Update the data with predictions generated via the current model (expected counts)
 - **Maximisation (M):**
Given the updated data, update the model parameters using MLE
 - Same step as when learning parameters for fully observable networks

EM: Maximisation Step (Refining Parameters)

- Now, we can refine parameters by applying ML using the expected counts

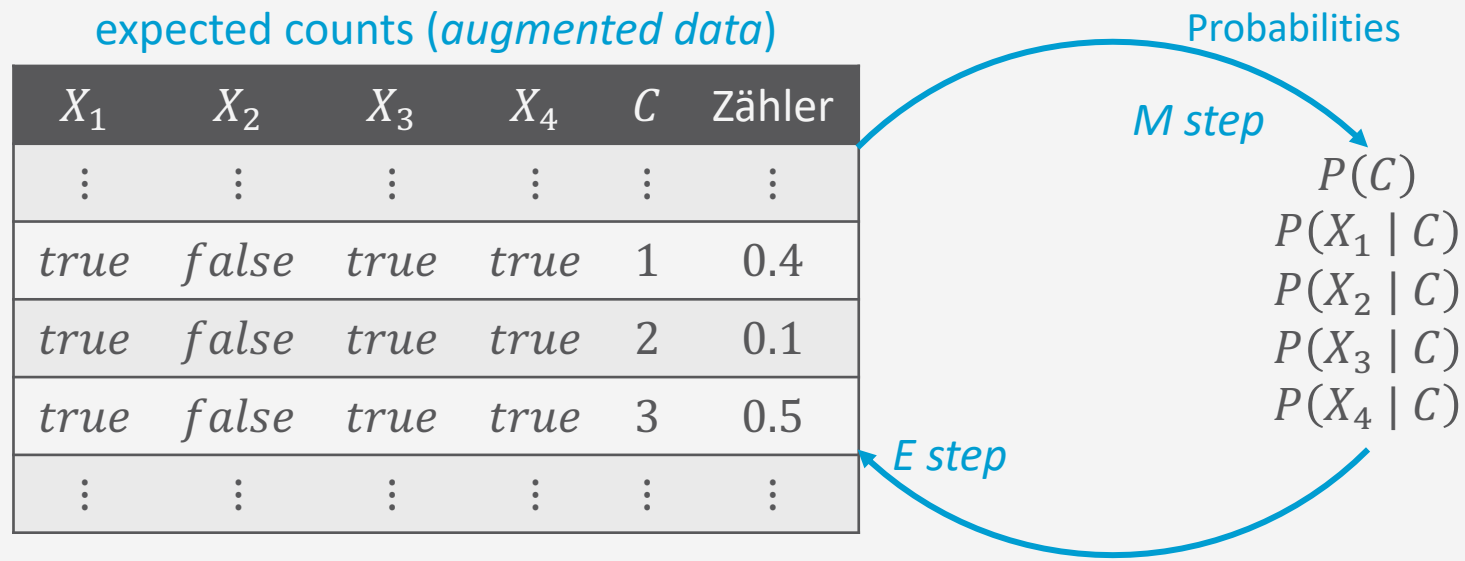
$$P(C = k) = \frac{\hat{N}(C = k)}{N}$$

$$P(X_i = x \mid C = k) = \frac{\hat{N}(X_i = x, C = k)}{\hat{N}(C = k)}$$

- for each $x \in \text{ran}(X_i)$, $i = 1, \dots, 4$, $k = 1, 2, 3$

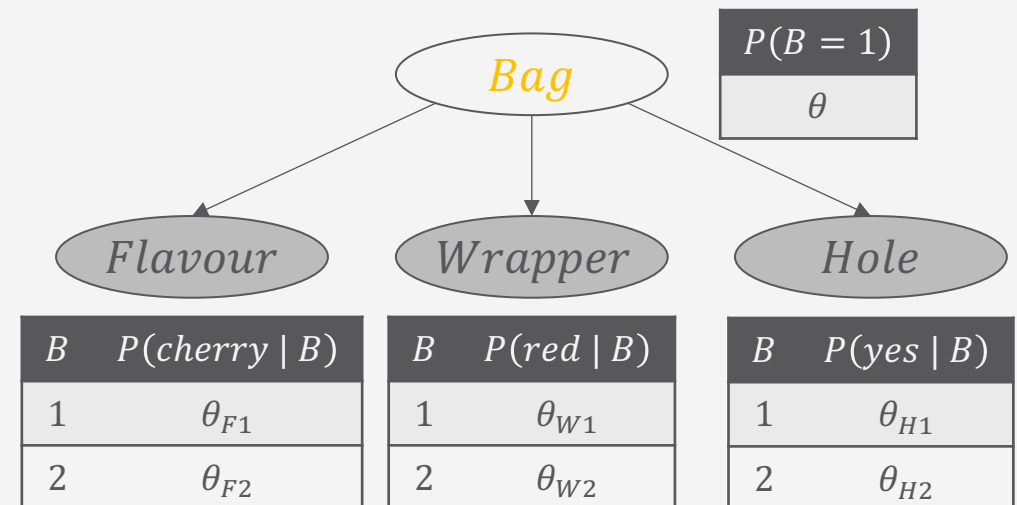
EM Cycle

- Now, the E step can be repeated
 - Calculate new expected counts given the just updated parameters
- Then, the M step can be repeated
 - Update parameters given the just newly computed expected counts



Example: Candy Factory Again

- Two bags of candy (1 and 2) have been mixed together
- Candies described by three features
 - *Flavour*
 - *Wrapper*
 - *Hole* (whether they have a hole in the middle)
 - Features depend probabilistically on the bag they originally came from
- For each candy, we want to predict, which was its original bag, from its features:
Naïve Bayes classifier

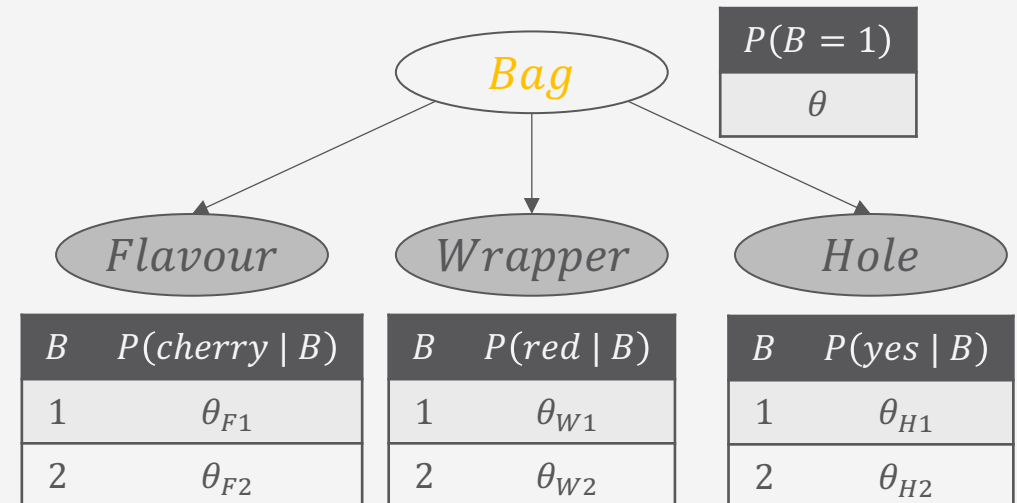


Data

- Assume the true parameters are
 - $\theta = 0.5$
 - $\theta_{F1} = \theta_{W1} = \theta_{H1} = 0.8$
 - $\theta_{F2} = \theta_{W2} = \theta_{H2} = 0.3$
- Assume that following counts have been sampled from $P(B, F, W, H)$, $N = 1000$

Data	$W = red$		$W = yellow$	
	$H = yes$	$H = no$	$H = yes$	$H = no$
$F = cherry$	273	93	104	90
$F = lime$	79	100	94	167

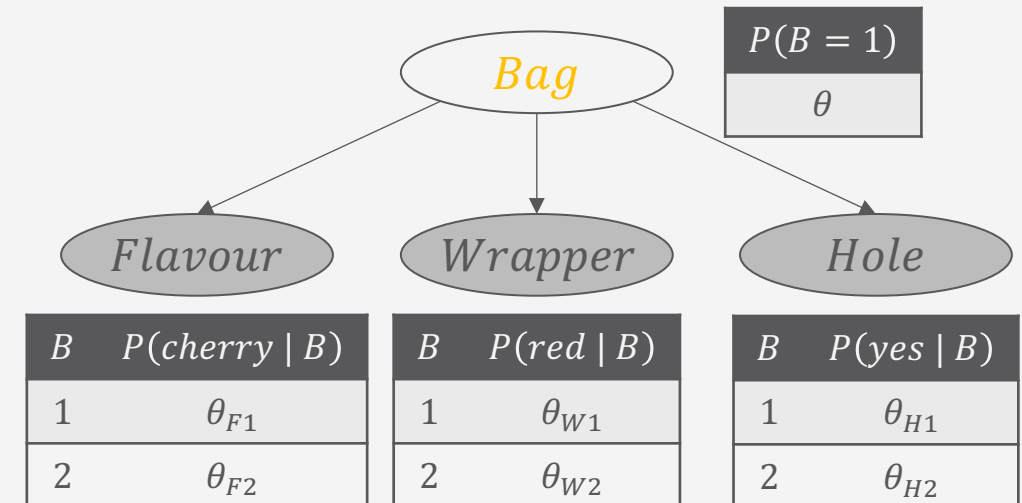
- We want to re-learn the true parameters using EM



EM: Initialisation

- Assign arbitrary initial parameters
 - Usually done randomly; here we select numbers convenient for computation
 - $\theta^{(0)} = 0.6$
 - $\theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6$
 - $\theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4$
- Work through one cycle of EM to compute $\theta^{(1)}$
 - We will also look at $\theta_{F1}^{(1)}$ afterwards
 - Conceptionally, one would first perform the E step for all and then the M step for all

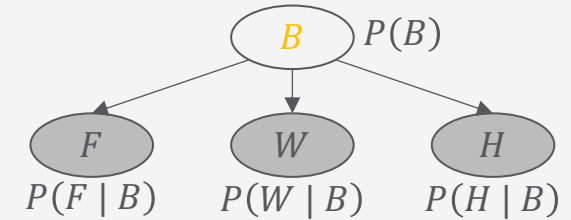
Data	$W = red$		$W = yellow$	
	$H = yes$	$H = no$	$H = yes$	$H = no$
$F = cherry$	273	93	104	90
$F = lime$	79	100	94	167



E Step

- First, we compute the expected count of candies from Bag 1:
 - Sum of the probabilities that each of the N data points comes from Bag 1
 - Let f_j, w_j, h_j be the values of the corresponding attributes for the j 'th datapoint

$$\begin{aligned}
 \hat{N}(\text{Bag} = 1) &= \sum_{j=1}^N P(\text{Bag} = 1 \mid f_j, w_j, h_j) \\
 &= \sum_{j=1}^N \frac{P(f_j, w_j, h_j \mid \text{Bag} = 1)P(\text{Bag} = 1)}{P(f_j, w_j, h_j)} \\
 &= \sum_{j=1}^N \frac{P(f_j \mid \text{Bag} = 1)P(w_j \mid \text{Bag} = 1)P(h_j \mid \text{Bag} = 1)P(\text{Bag} = 1)}{\sum_{k=1}^2 P(f_j \mid \text{Bag} = k)P(w_j \mid \text{Bag} = k)P(h_j \mid \text{Bag} = k)P(\text{Bag} = k)}
 \end{aligned}$$



E Step

$$\hat{N}(Bag = 1) = \sum_{j=1}^N \frac{P(f_j | Bag = 1)P(w_j | Bag = 1)P(h_j | Bag = 1)P(Bag = 1)}{\sum_{k=1}^2 P(f_j | Bag = k)P(w_j | Bag = k)P(h_j | Bag = k)P(Bag = k)}$$

- Summation can be broken down into the 8 candy groups in the data table
 - For instance, the sum over the 273 cherry candies with read wrapper and hole (first entry in the data table) gives

$$273 \frac{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)}}{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)} + \theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} (1 - \theta^{(0)})}$$

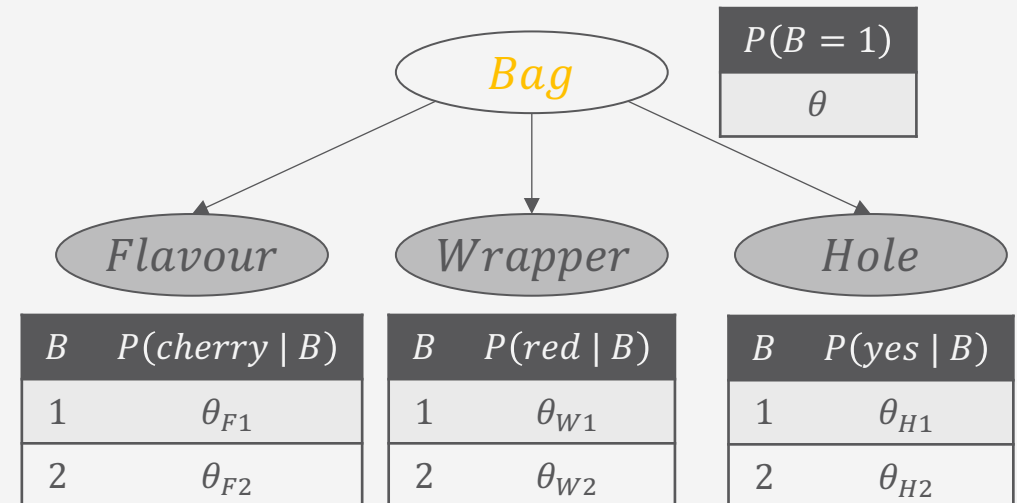
$$= 273 \frac{0.6^4}{0.6^4 + 0.4^4} = 273 \frac{0.1296}{0.1552} = 227.97$$

$$\theta^{(0)} = 0.6$$

$$\theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6$$

$$\theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4$$

Data	W = red		W = yellow	
	H = yes	H = no	H = yes	H = no
F = cherry	273	93	104	90
F = lime	79	100	94	167



M Step

- If we do compute the sums over the other 7 candy groups, we get

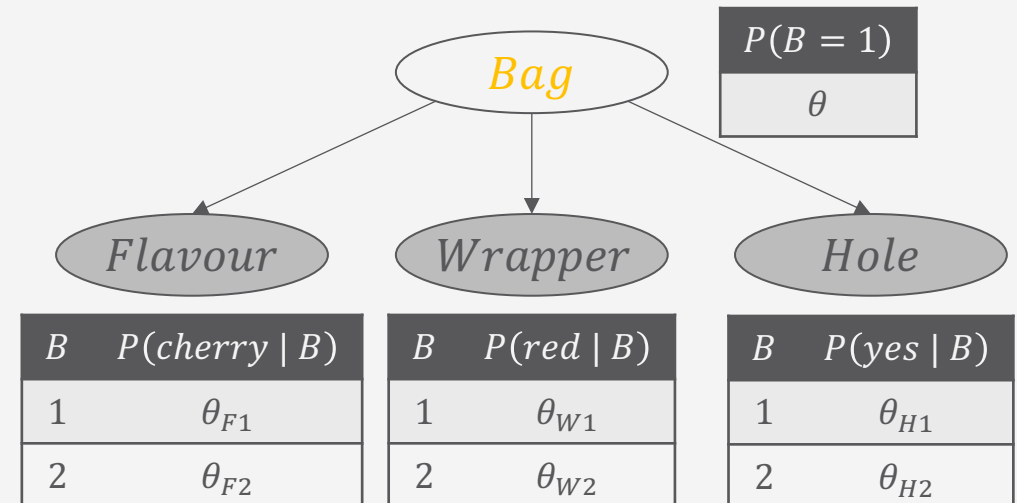
$$\hat{N}(Bag = 1) = 612.4$$

- At this point, we can perform the M step to refine θ by taking the expected frequency of the data points that come from Bag 1
 - Expected count of Bag 1 divided by N

$$\theta^{(1)} = \frac{\hat{N}(Bag = 1)}{N} = \frac{612.4}{1000} = 0.6124$$

$$\begin{aligned}\theta^{(0)} &= 0.6 \\ \theta_{F1}^{(0)} &= \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6 \\ \theta_{F2}^{(0)} &= \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4\end{aligned}$$

Data	$W = red$		$W = yellow$	
	$H = yes$	$H = no$	$H = yes$	$H = no$
$F = cherry$	273	93	104	90
$F = lime$	79	100	94	167



Another Parameter: θ_{F1}

- E step: Compute expected count of cherry candies from Bag 1 given $\theta^{(0)}$

$$\hat{N}(B = 1, F = \text{cherry}) = \sum_{j:F=\text{cherry}} P(B = 1 \mid \text{cherry}_j, w_j, h_j)$$

- Same as before:

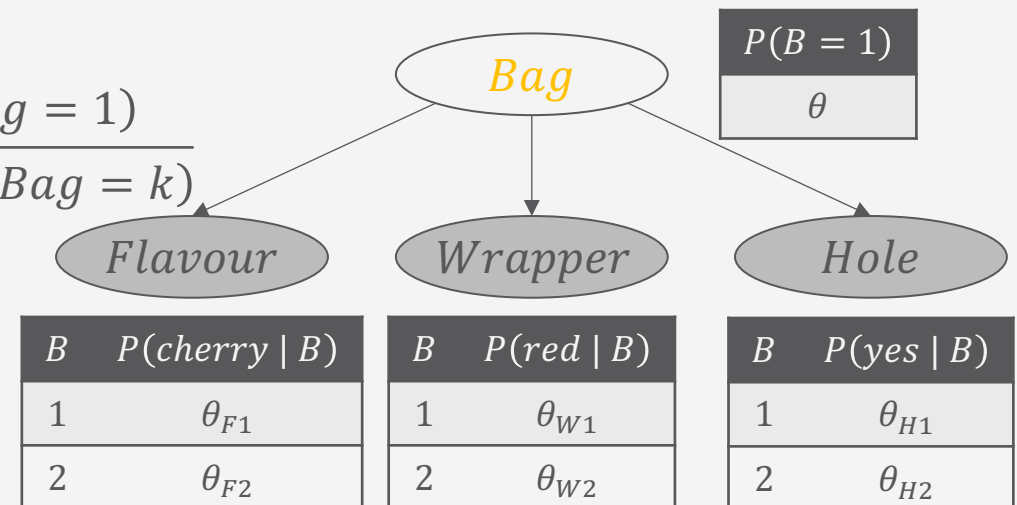
$$\begin{aligned} & \hat{N}(\text{Bag} = 1, F = \text{cherry}) \\ &= \sum_{j=1}^N \frac{P(\text{cherry}_j \mid \text{Bag} = 1)P(w_j \mid \text{Bag} = 1)P(h_j \mid \text{Bag} = 1)P(\text{Bag} = 1)}{\sum_{k=1}^2 P(\text{cherry}_j \mid \text{Bag} = k)P(w_j \mid \text{Bag} = k)P(h_j \mid \text{Bag} = k)P(\text{Bag} = k)} \end{aligned}$$

- M step: Refine θ_{F1} by computing relative frequencies

$$\theta_{F1}^{(1)} = \frac{\hat{N}(B = 1, F = \text{cherry})}{\hat{N}(B = 1)}$$

$$\begin{aligned} \theta^{(0)} &= 0.6 \\ \theta_{F1}^{(0)} &= \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6 \\ \theta_{F2}^{(0)} &= \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4 \end{aligned}$$

Data	$W = \text{red}$		$W = \text{yellow}$	
	$H = \text{yes}$	$H = \text{no}$	$H = \text{yes}$	$H = \text{no}$
$F = \text{cherry}$	273	93	104	90
$F = \text{lime}$	79	100	94	167



Learning Result

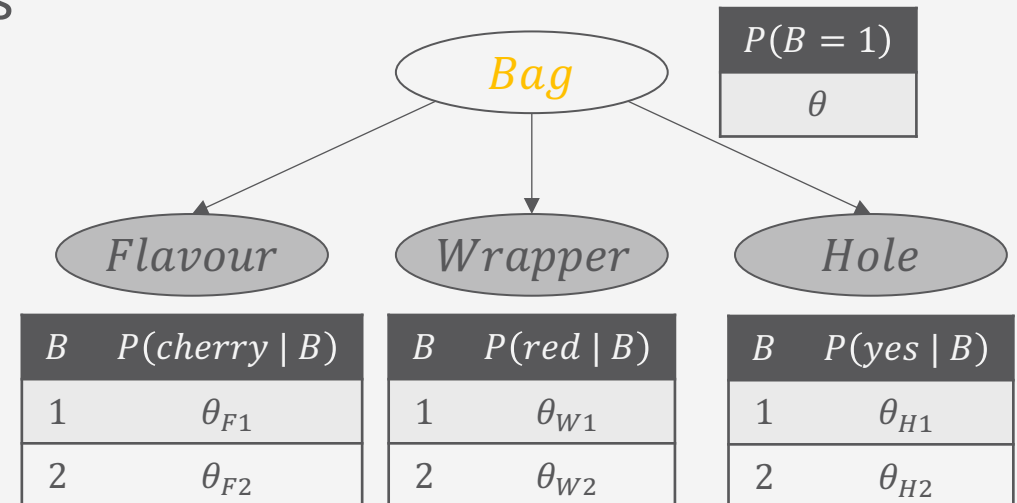
- After a full cycle through all parameters:
 - $\theta^{(1)} = 0.6124$
 - $\theta_{F1}^{(1)} = 0.6684, \theta_{W1}^{(1)} = 0.6483, \theta_{H1}^{(1)} = 0.658$
 - $\theta_{F2}^{(1)} = 0.3887, \theta_{W2}^{(1)} = 0.3817, \theta_{H2}^{(1)} = 0.3827$
- For any set of parameters, compute log likelihood as we did before

$$\theta^{(0)} = 0.6$$

$$\theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6$$

$$\theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4$$

Data	$W = red$		$W = yellow$	
	$H = yes$	$H = no$	$H = yes$	$H = no$
$F = cherry$	273	93	104	90
$F = lime$	79	100	94	167



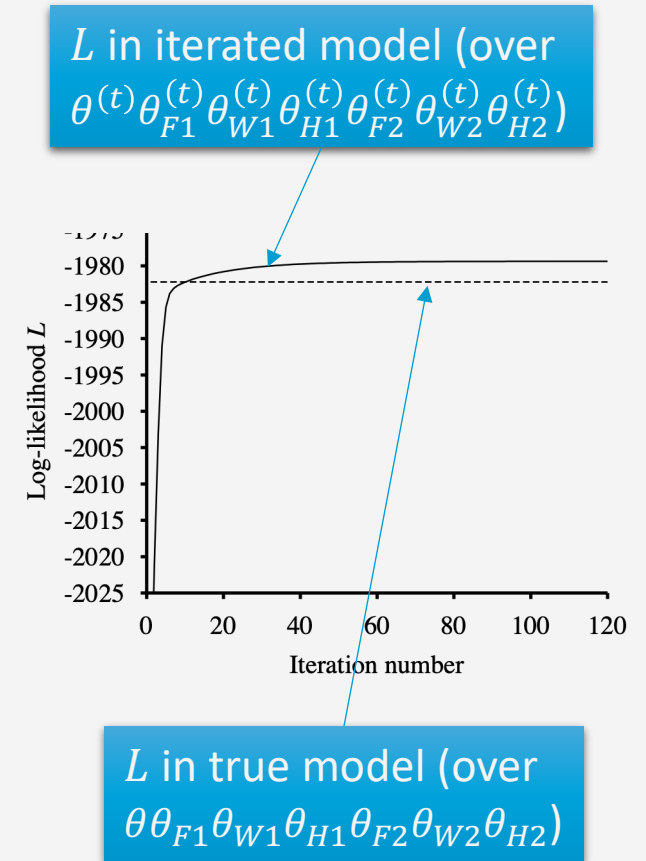
Learning Result

- Log likelihood of parameters:

$$P(\mathbf{d} | h_{\theta^{(t)} \theta_{F1}^{(t)} \theta_{W1}^{(t)} \theta_{H1}^{(t)} \theta_{F2}^{(t)} \theta_{W2}^{(t)} \theta_{H2}^{(t)}}) = \prod_{j=1}^N P(d_j | h_{\theta^{(t)} \theta_{F1}^{(t)} \theta_{W1}^{(t)} \theta_{H1}^{(t)} \theta_{F2}^{(t)} \theta_{W2}^{(t)} \theta_{H2}^{(t)}})$$

$$\Rightarrow \log P(\mathbf{d}, h_{\theta}) = \sum_{j=1}^N \log P(d_j | h_{\theta^{(t)} \theta_{F1}^{(t)} \theta_{W1}^{(t)} \theta_{H1}^{(t)} \theta_{F2}^{(t)} \theta_{W2}^{(t)} \theta_{H2}^{(t)}})$$

- One can show that this value increases with each EM cycle (**convergence**)
 - Example candy factory: Log likelihood $L = \log P(\mathbf{d}, h_{\theta})$ increases with each iteration [figure right]
- EM with ML can get stuck in **local maxima**
 - Possible fix: start with different initial values, use MAP learning



EM: Discussion

- For more complex BNs, the algorithm is basically the same
 - In general, compute CPD entries for each random variable R_i given its parent values $\text{pa}(R_i)$

$$\theta_{ijk} = P(R_i = r_{ij} \mid \text{Pa}(R_i) = \text{pa}_k)$$

$$\theta_{ijk} = \frac{\hat{N}(R_i = r_{ij}, \text{Pa}(R_i) = \text{pa}_k)}{\hat{N}(\text{Pa}(R_i) = \text{pa}_k)}$$

- Expected counts computed by summing over samples, after having computed all necessary $P(R_i = r_{ij}, \text{Pa}(R_i) = \text{pa}_{ik})$ using any BN inference algorithm (e.g., VE, JT, sampling-based)
- Inference can be intractable
 - Fix: Use sampling algorithms for E step

Undirected Models

- BNs have the advantage of a normalisation constant of $Z = 1$
 - Parameter estimation reduces to estimating parameters of local CPTs
 - Undirected models do not have this advantage

$$P_F = \frac{1}{Z} \prod_{i=1}^n \phi_i(R_1, \dots, R_k)$$
$$Z = \sum_{r_1 \in \text{ran}(R_1)} \sum_{r_n \in \text{ran}(R_n)} \prod_{i=1}^k \phi_i(r_1, \dots, r_k)$$

- Z combines all variables in model in one function

ML Procedure for Undirected Models

- Given a model $F = \{f_i\}_{i=1}^n$ with l random variables $R \in \text{rv}(F)$ with range values $r \in \text{ran}(R)$
 - Let $W = \times_{i=1}^l \text{ran}(R_i)$ denote the set of possible worlds
 - With $\mathbf{r} = (r_1, \dots, r_l)$ referring to a single world (compound event for $\text{rv}(F)$)
 - Let $\mathbf{r}_\alpha$ denote the projection of $\mathbf{r}$ onto the random variables of some entity α
 - E.g., $\mathbf{r}_f = \pi_{\text{rv}(f)}(\mathbf{r})$: project $\mathbf{r}$ onto the random variables in f
 - Let ϕ_f refer to the potential function of f
- Given a data set D with k compound events for $\text{rv}(F)$ (i.e., complete data)
 - Let $\#(\mathbf{r})$ denote how often $\mathbf{r}$ has been observed in D
 - Could write D as multi-set, i.e., $D = \left\{(\mathbf{r}, \#(\mathbf{r}))\right\}_{i=1}^m$

ML Procedure for Undirected Models

1. Express the likelihood $P(D|\theta)$ of the data D as a function of the parameters θ to be learned

- θ refers to the potentials in the factors of F

$$P(D|\theta) = \prod_{\mathbf{r} \in W} P(\mathbf{r}|\theta)^{\#(\mathbf{r})}$$

- In this formulation, $P(D|\theta)$ can become zero if an $\mathbf{r} \in W$ has not been observed
 - E.g., initialise all counts to 1 to circumvent problem
- and take the logarithm

$$\log P(D|\theta) = \log \prod_{\mathbf{r} \in W} P(\mathbf{r}|\theta)^{\#(\mathbf{r})}$$

ML Procedure for Undirected Models

$$\begin{aligned}
 \log P(D|\theta) &= \log \prod_{\mathbf{r} \in W} P(\mathbf{r}|\theta)^{\#(\mathbf{r})} && \phi_f(\mathbf{r}_f) \text{ refers to parameter } \in \theta \\
 &= \sum_{\mathbf{r} \in W} \#(\mathbf{r}) \log P(\mathbf{r}|\theta) = \sum_{\mathbf{r} \in W} \#(\mathbf{r}) \log \left(\frac{1}{Z} \prod_{f \in F} \phi_f(\mathbf{r}_f) \right) \\
 &= \sum_{\mathbf{r} \in W} \#(\mathbf{r}) \left(\log \left(\frac{1}{Z} \right) + \log \prod_{f \in F} \phi_f(\mathbf{r}_f) \right) \\
 &= \sum_{\mathbf{r} \in W} \#(\mathbf{r}) \left(-\log(Z) + \sum_{f \in F} \log \phi_f(\mathbf{r}_f) \right) \\
 &= \left(\sum_{\mathbf{r} \in W} \#(\mathbf{r}) \sum_{f \in F} \log \phi_f(\mathbf{r}_f) \right) - \sum_{\mathbf{r} \in W} \#(\mathbf{r}) \log(Z) \\
 &= \left(\sum_{\mathbf{r} \in W} \#(\mathbf{r}) \sum_{f \in F} \log \phi_f(\mathbf{r}_f) \right) - k \log(Z) \\
 &= \left(\sum_{f \in F} \sum_{\mathbf{r}_f \in \text{ran}(\text{rv}(f))} \#(\mathbf{r}_f) \log \phi_f(\mathbf{r}_f) \right) - k \log(Z)
 \end{aligned}$$

ML Procedure for Undirected Models

2. Take the derivative of the log likelihood with respect to each parameter, i.e., $\phi_f(\mathbf{r}_f)$

$$\log P(D|\theta) = \left(\sum_{f \in F} \sum_{\mathbf{r}_f \in \text{ran}(\text{rv}(f))} \underbrace{\#(\mathbf{r}_f) \log \phi_f(\mathbf{r}_f)}_{l_1} \right) - \underbrace{k \log(Z)}_{l_2}$$

- Derivation of l_1

$$\frac{\partial l_1}{\partial \phi_f(\mathbf{r}_f)} = \#(\mathbf{r}_f) \cdot \frac{1}{\phi_f(\mathbf{r}_f)}$$

constant w.r.t. $\phi_f(\mathbf{r}_f)$

Derivation of log:

$$\frac{\partial \log f(x)}{\partial x} = \frac{1}{f(x)} \cdot \frac{\partial f(x)}{\partial x}$$

E.g., if $f(x) = x$:

$$\frac{\partial \log x}{\partial x} = \frac{1}{x} \cdot \frac{\partial x}{\partial x} = \frac{1}{x} \cdot 1 = \frac{1}{x}$$

ML Procedure for Undirected Models

- Derivation of $l_2 = k \log(Z)$

$$\frac{\partial l_2}{\partial \phi_f(\mathbf{r}_f)} = k \cdot \frac{1}{Z} \cdot \frac{\partial Z}{\partial \phi_f(\mathbf{r}_f)} = k \cdot \frac{1}{Z} \cdot \underbrace{\frac{\partial}{\partial \phi_f(\mathbf{r}_f)} \cdot \sum_{s \in W} \prod_{e \in F} \phi_e(\mathbf{s}_e)}_{\text{blue bracket}} = k \cdot \frac{1}{Z} \cdot \sum_{s \in W} \underbrace{\delta(\mathbf{r}_f, \mathbf{s}_f) \frac{\partial}{\partial \phi_f(\mathbf{r}_f)} \cdot \prod_{e \in F} \phi_e(\mathbf{s}_e)}_{\text{yellow bracket}}$$

Only those summands that contain $\phi_f(\mathbf{r}_f)$ remain; derivation of the others returns 0; use indicator function to denote this:

$$\delta(\mathbf{r}, \mathbf{s}) = \begin{cases} 1 & \text{if } \mathbf{r} = \mathbf{s} \\ 0 & \text{otherwise} \end{cases}$$

One of the e 's is f , i.e., $\phi_f(\mathbf{r}_f) = \phi_e(\mathbf{s}_e)$ for one e ; the rest is constant w.r.t. $\phi_f(\mathbf{r}_f)$

$$= \frac{\partial}{\partial \phi_f(\mathbf{r}_f)} \cdot \phi_f(\mathbf{s}_f) \cdot \prod_{\substack{e \in F \\ e \neq f}} \phi_e(\mathbf{s}_e) = \prod_{\substack{e \in F \\ e \neq f}} \phi_e(\mathbf{s}_e)$$

$$= \frac{\phi_f(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} \cdot \prod_{\substack{e \in F \\ e \neq f}} \phi_e(\mathbf{s}_e) = \frac{1}{\phi_f(\mathbf{r}_f)} \cdot \prod_{e \in F} \phi_e(\mathbf{s}_e)$$

ML Procedure for Undirected Models

- Derivation of l_2

$$\begin{aligned}\frac{\partial l_2}{\partial \phi_f(\mathbf{r}_f)} &= k \cdot \frac{1}{Z} \cdot \sum_{\mathbf{s} \in W} \delta(\mathbf{r}_f, \mathbf{s}_f) \frac{1}{\phi_f(\mathbf{r}_f)} \cdot \prod_{e \in F} \phi_e(\mathbf{s}_e) \\ &= k \cdot \frac{1}{\phi_f(\mathbf{r}_f)} \cdot \sum_{\mathbf{s} \in W} \delta(\mathbf{r}_f, \mathbf{s}_f) \cdot \frac{1}{Z} \cdot \prod_{e \in F} \phi_e(\mathbf{s}_e) \\ &= k \cdot \frac{1}{\phi_f(\mathbf{r}_f)} \cdot \sum_{\mathbf{s} \in W} \delta(\mathbf{r}_f, \mathbf{s}_f) \cdot P(\mathbf{s}) \\ &= k \cdot \frac{1}{\phi_f(\mathbf{r}_f)} \cdot P(\mathbf{r}_f)\end{aligned}$$

ML Procedure for Undirected Models

- Take the derivative of the log likelihood with respect to each parameter, i.e., $\phi_f(\mathbf{r}_f)$

$$\log P(D|\theta) = \left(\sum_{f \in F} \sum_{\mathbf{r}_f \in \text{ran}(\text{rv}(f))} \underbrace{\#(\mathbf{r}_f) \log \phi_f(\mathbf{r}_f)}_{l_1} \right) - \underbrace{k \log(Z)}_{l_2}$$

- Derivative for each parameter $\phi_f(\mathbf{r}_f)$

$$\frac{\partial l_1}{\partial \phi_f(\mathbf{r}_f)} = \frac{\#(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)}$$

$$\frac{\partial l_2}{\partial \phi_f(\mathbf{r}_f)} = k \cdot \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)}$$

$$\frac{\partial \log P(D|\theta)}{\partial \phi_f(\mathbf{r}_f)} = \frac{\#(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} - k \cdot \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)}$$

ML Procedure for Undirected Models

3. Find the parameter value that makes the derivative equal to 0

$$\begin{aligned}\frac{\partial \log P(D|\theta)}{\partial \phi_f(\mathbf{r}_f)} &= \frac{\#(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} - k \cdot \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} = 0 \quad \rightarrow \quad \frac{\#(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} = k \cdot \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} \\ &\#(\mathbf{r}_f) = k \cdot P(\mathbf{r}_f) \\ &\frac{\#(\mathbf{r}_f)}{k} = P(\mathbf{r}_f)\end{aligned}$$

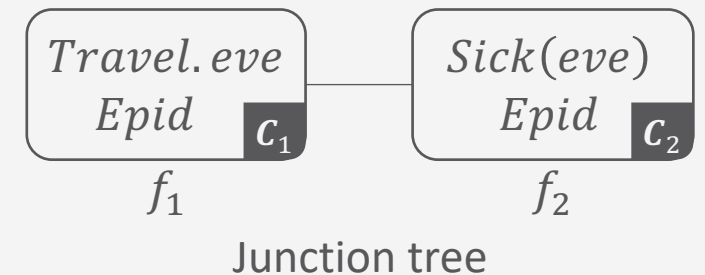
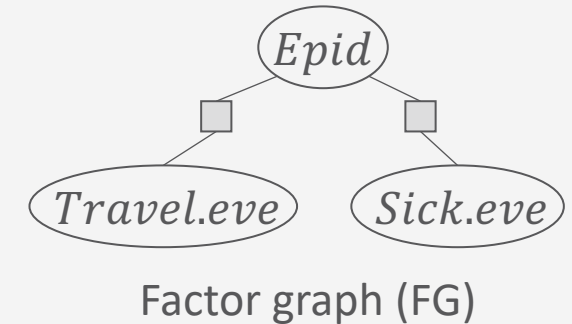
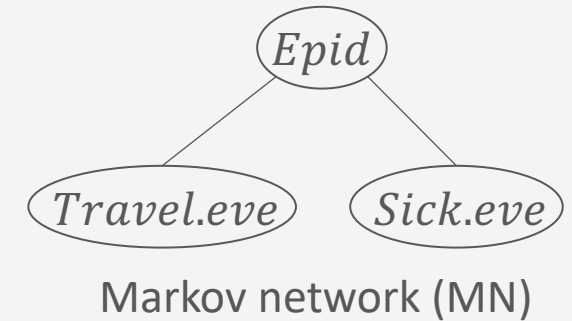
- States that the ML estimates should be in such a way that the model marginals $P(\mathbf{r}_f)$ are equal to the normalised empirical counts:

$$P^\#(\mathbf{r}_f) \stackrel{\text{def}}{=} \frac{\#(\mathbf{r}_f)}{k} \stackrel{!}{=} P(\mathbf{r}_f)$$

- Does not state how to get the estimates

Factors over Maximal Cliques

- Markov net with potential functions over its maximal cliques
 - Clique: set of nodes where every node is directly connected with every other node
 - In factor graphs: Directly connected = appear in same factor
 - Maximal clique
 - There is no node in the graph that you could add to the clique with the clique remaining a clique
 - Equivalent factor models
- Corresponding jtree with one factor per cluster assigned as local model



Factors over Maximal Cliques

- Factor model $F = \{f_i\}_{i=1}^n$ with a junction tree (V, E) where each C_i has one factor f_i assigned
 - Like in Hugin
- Full joint represented:

$$P_F = \frac{1}{Z} \prod_{f \in F} f = \frac{1}{Z} \prod_{C_i \in V} f_i = \frac{\prod_{C_i \in V} P(C_i)}{\prod_{\{i,j\} \in E} P(s_{ij})}$$

- For a world $\mathbf{r}$,

$$P(\mathbf{r}) = \frac{\prod_{C_i \in V} P(\mathbf{r}_i)}{\prod_{\{i,j\} \in E} P(\mathbf{r}_{ij})} = \frac{\prod_{f \in F} P(\mathbf{r}_f)}{\prod_{\{i,j\} \in E} P(\mathbf{r}_{ij})}$$

Factors over Maximal Cliques

$$P(\mathbf{r}) = \frac{\prod_{C_i \in V} P(\mathbf{r}_i)}{\prod_{\{i,j\} \in E} P(\mathbf{r}_{ij})} = \frac{\prod_{f \in F} P(\mathbf{r}_f)}{\prod_{\{i,j\} \in E} P(\mathbf{r}_{ij})}$$

$$P^\#(\mathbf{r}_f) \stackrel{\text{def}}{=} \frac{\#(\mathbf{r}_f)}{k} \stackrel{!}{=} P(\mathbf{r}_f)$$

- Set factors to be the normalised empirical counts $P^\#(\mathbf{r}_f)$ and, for each separator, pick one neighbour and divide the factor by the normalised empirical count of the separator

$P^\#(\mathbf{r}_{ij})$, i.e.,

- For each $f \in F$,
 - Set $\phi_f(\mathbf{r}_f) \leftarrow P^\#(\mathbf{r}_f)$
- For each $\{i,j\} \in E$, i.e., separator S_{ij} ,
 - Choose $h \in \{i,j\}$ at random
 - Set $\phi_h(\mathbf{r}_h) \leftarrow \frac{\phi_h(\mathbf{r}_h)}{P^\#(\mathbf{r}_{ij})}$

LearnMaxCliques

- Enforces $Z = 1$ as we have now probability distributions in the factors

Learning with General Factors

- If factors over non-maximal cliques, closed form solution not possible; fixed-point iteration:

$$\frac{\partial \log P(D|\theta)}{\partial \phi_f(\mathbf{r}_f)} = \frac{\#(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} - k \cdot \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} = 0 \quad \rightarrow \quad \frac{\#(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} = k \cdot \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)}$$

$$\frac{\#(\mathbf{r}_f)}{k \cdot \phi_f(\mathbf{r}_f)} = \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)}$$

$$\frac{P^\#(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)} = \frac{P(\mathbf{r}_f)}{\phi_f(\mathbf{r}_f)}$$

$$\frac{\phi_f(\mathbf{r}_f)}{P^\#(\mathbf{r}_f)} = \frac{\phi_f(\mathbf{r}_f)}{P(\mathbf{r}_f)}$$

$$\phi_f(\mathbf{r}_f) = \phi_f(\mathbf{r}_f) \frac{P^\#(\mathbf{r}_f)}{P(\mathbf{r}_f)}$$

- Update rule

$$\phi_f^{(t+1)}(\mathbf{r}_f) \leftarrow \phi_f^t(\mathbf{r}_f) \frac{P^\#(\mathbf{r}_f)}{P^t(\mathbf{r}_f)}$$

- $\phi_f^t(\mathbf{r}_f)$ current potentials
- $P^\#(\mathbf{r}_f)$ fixed
- $P^t(\mathbf{r}_f)$ query to compute on current $\phi_f^t(\mathbf{r}_f)$

Iterative proportional fitting (IPF)

- Initialise all factors with $t = 0$ uniformly, e.g., all potentials 1
- **for** $t = 1, 2, \dots$ **do**
 - **if** convergence criterion does not hold **then**
 - **for** all $\mathbf{r}_f \in \text{ran}(\text{rv}(f)), f \in F$ **do**
 - $\phi_f^{(t+1)}(\mathbf{r}_f) \leftarrow \phi_f^t(\mathbf{r}_f) \frac{p^\#(\mathbf{r}_f)}{p^t(\mathbf{r}_f)}$
 - **else break**
- Convergence criterion, e.g., given error threshold ε
 - $\forall f \in F : \phi_f^{(t+1)}(\mathbf{r}_f) - \phi_f^t(\mathbf{r}_f) < \varepsilon$
- Many $p^t(\mathbf{r}_f)$ queries to compute over all factors!
 - Efficient execution using a junction tree (jtree)
 - Jtree construction independent of parameters

IPF

IPF with Jtrees

- Construct a jtree
- Initialise all factors and messages uniformly for $t = 0$
- Pick a random cluster $\mathcal{C}_i$ as the current cluster
- **for** $t = 1, 2, \dots$ **do**
 - **if** convergence criterion does not hold **then**
 - **for** all $\mathbf{r}_f \in \text{ran}(\text{rv}(f)), f \in F_i^t$ **do**
 - $\phi_f^{(t+1)}(\mathbf{r}_f) \leftarrow \phi_f^t(\mathbf{r}_f) \frac{P^\#(\mathbf{r}_f)}{P^t(\mathbf{r}_f)}$
 - **else break**
 - Choose a neighbour $\mathcal{C}_j$ as new current cluster at random
 - Compute and send message m_{ij}^t to $\mathcal{C}_j$

Compute $P^t(\mathbf{r}_f)$ using current m_{ji}^t and F_i^t

- Organise in a reasonable way over all local factors and assignments

No ordering prescribed; implicitly required that all $\mathcal{C}_i$ visited enough

- E.g., start at a leaf, traverse the clusters by depth first search

IPF-JT

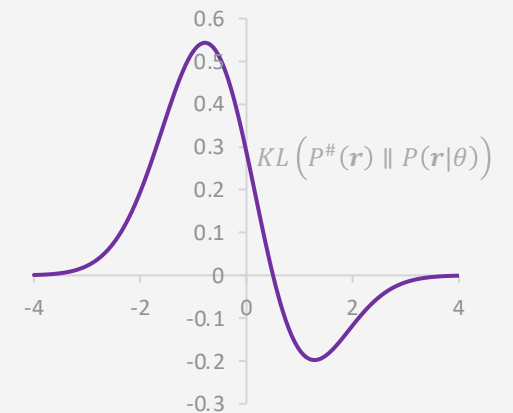
Properties of IPF Updates

- Reduces to one update per factor if factors are over maximal cliques
- Coordinate ascent algorithm
 - Coordinates = parameters θ in clusters
 - At each step, the update increases the log-likelihood of the data $\log P(D|\theta)$ and it will converge to a global maximum
- Maximising the log-likelihood is equivalent to minimising the KL divergence (cross entropy)

$$\max \log P(D|\theta) \Leftrightarrow \min KL(P^\#(\mathbf{r}) \parallel P(\mathbf{r}|\theta))$$

$$KL(P^\#(\mathbf{r}) \parallel P(\mathbf{r}|\theta)) = \sum_{\mathbf{r}} P^\#(\mathbf{r}) \log \frac{P^\#(\mathbf{r})}{P(\mathbf{r}|\theta)}$$

- Max-entropy principle for parameterisation: dual perspective to MLE



IPF with Incomplete Data

- Problem with incomplete data:
 - Update rule

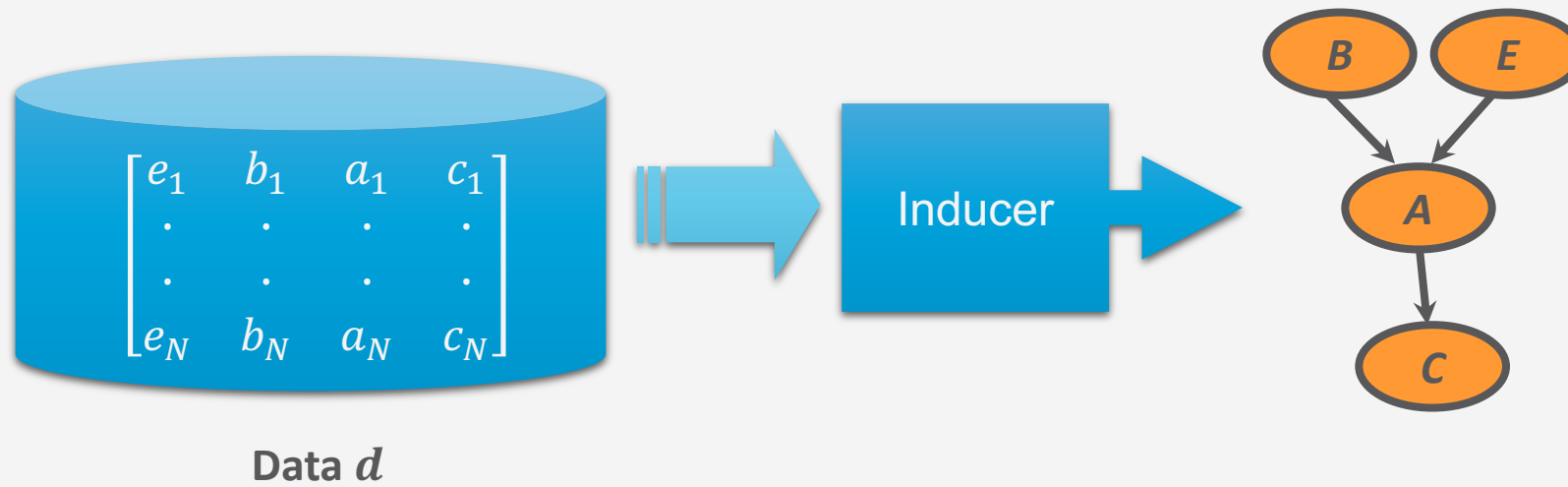
$$\phi_f^{(t+1)}(\mathbf{r}_f) \leftarrow \phi_f^t(\mathbf{r}_f) \frac{P^\#(\mathbf{r}_f)}{P^t(\mathbf{r}_f)}$$

- Hidden variables do not have $P^\#(\mathbf{r}_f)$
- **EM-IPF**: IPF procedure merged with EM scheme
 - **E** step: Calculate *expected* counts for hidden variables
 - **M** step: Update parameters $\phi_f^{(t+1)}(\mathbf{r}_f)$
 - Could also use jtree for efficiency

Structure Learning

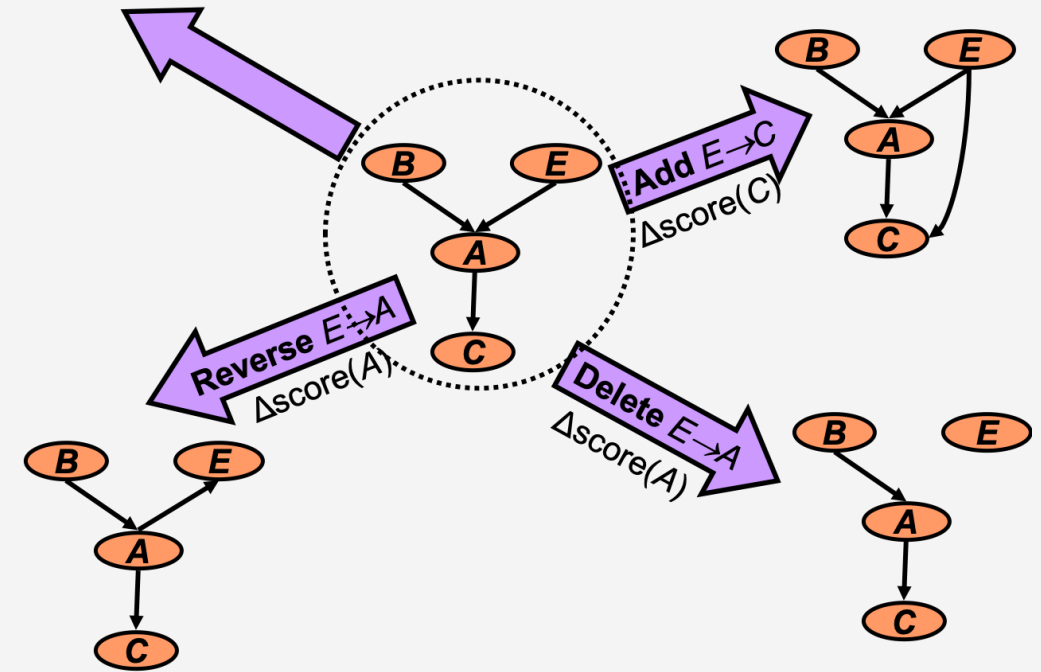
- Unknown network structure
- Given training set D
- Find model that best matches D , includes:
 - *Model selection*
 - Parameter estimation

Approaches sketched here are general frameworks that apply to structure learning in all sorts of PGMs



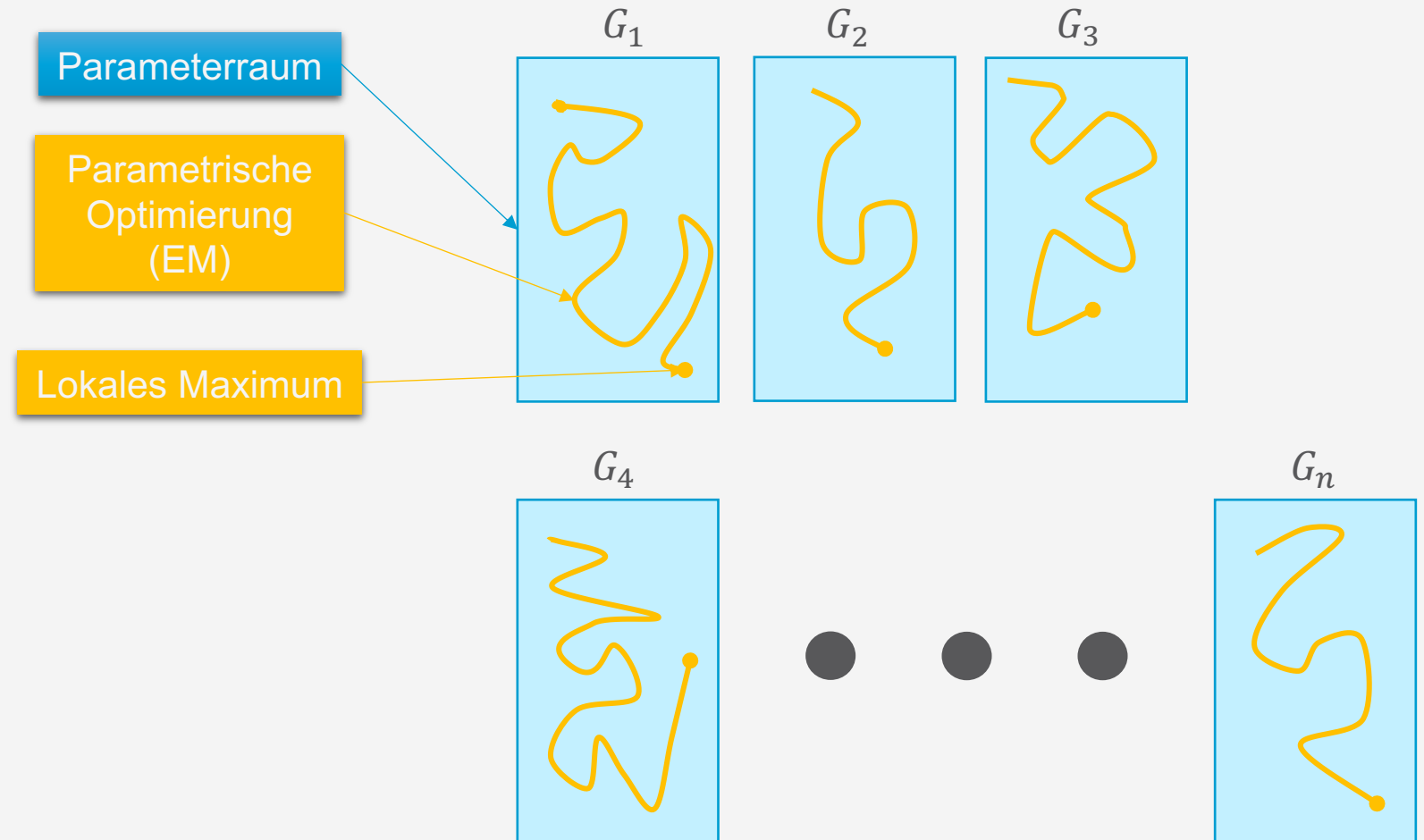
Model selection

- Goal: Select the best network structure
 - Input: Training data, scoring function
 - Output: A network that maximises the score
1. Perform heuristic search for model candidates
 2. Perform EM for parameters
 - If complete data
 - all variables known
 - MLE instead of EM
 3. Score each model
 4. Pick model with highest score



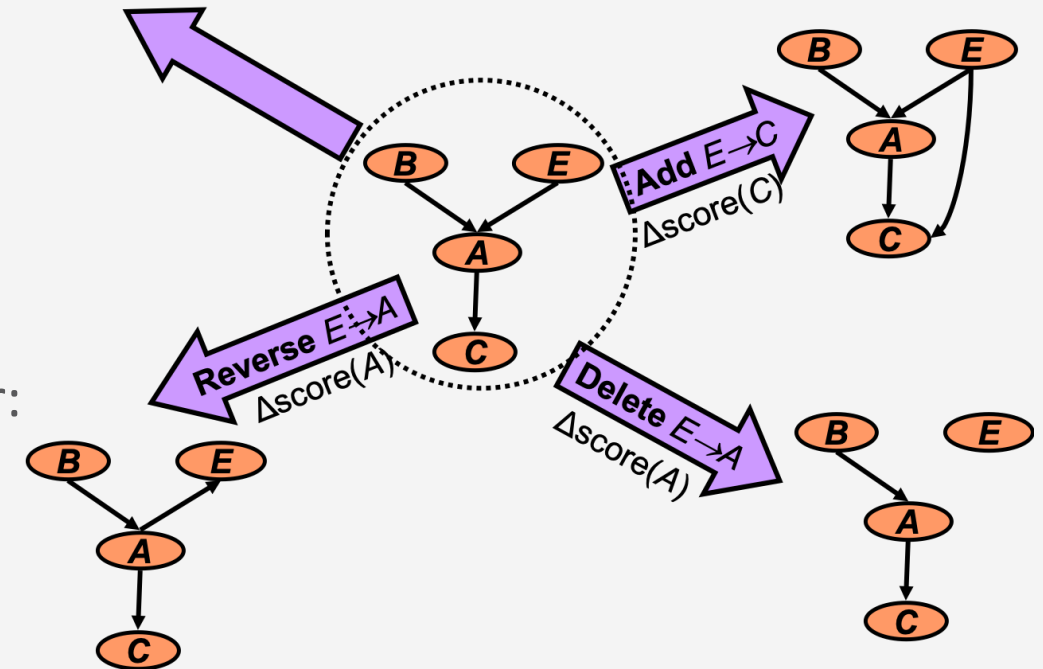
Local Search in Practice

- Perform EM for each candidate graph
- Computationally expensive
 - Parameter optimisation via EM – non-trivial
 - Need to perform EM for all candidate structures
 - Spend time even on poor candidates
 - Data might be incomplete



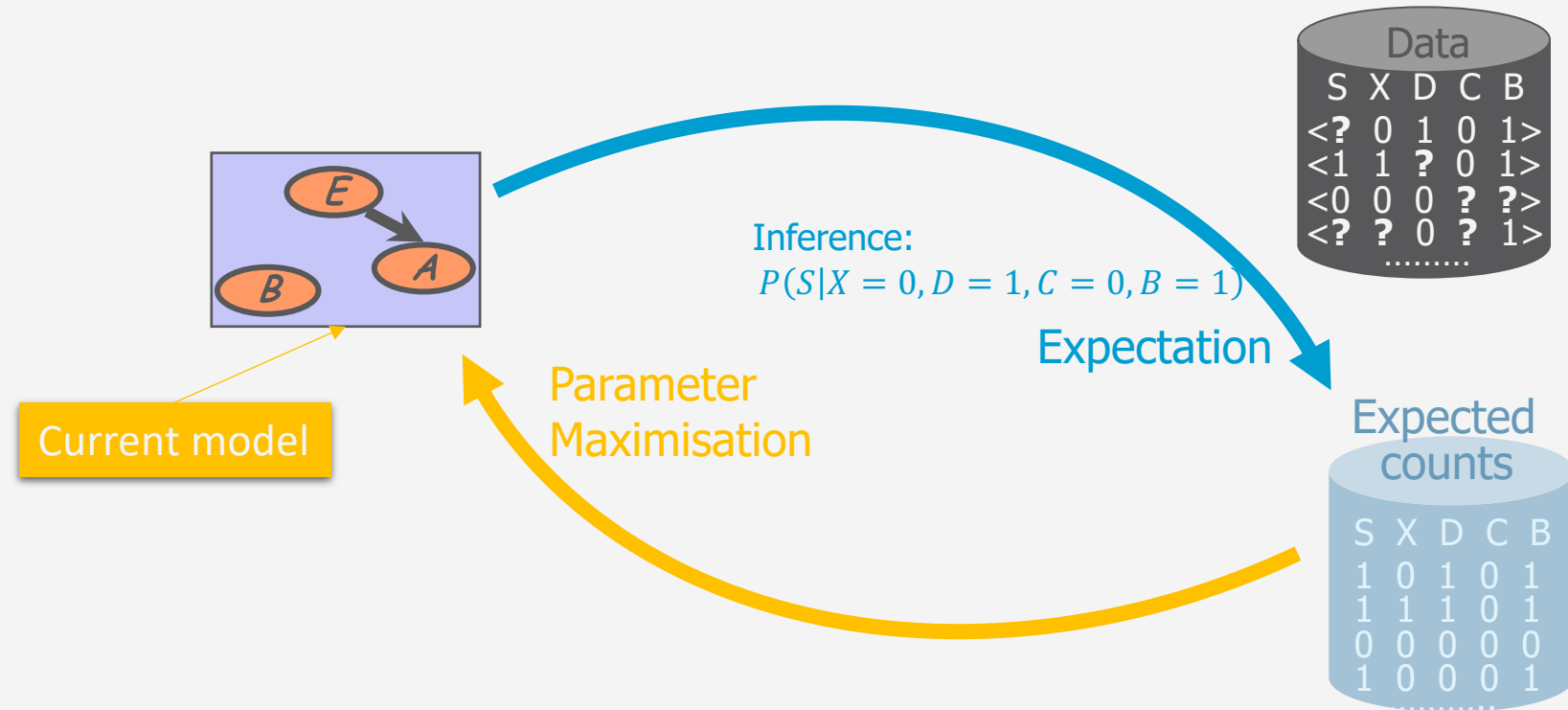
Structure Learning: Incomplete Data

- There might be hidden variables
 - No data available for the hidden variables
 - Search space becomes that much larger
- Idea:
 - Use current model to help evaluate new structures
- Outline:
 - Perform search in (Structure, Parameters) space
 - At each iteration, use current model for finding either:
 - Better scoring parameters: “parametric” EM step
or
 - Better scoring structure: “structural” EM step



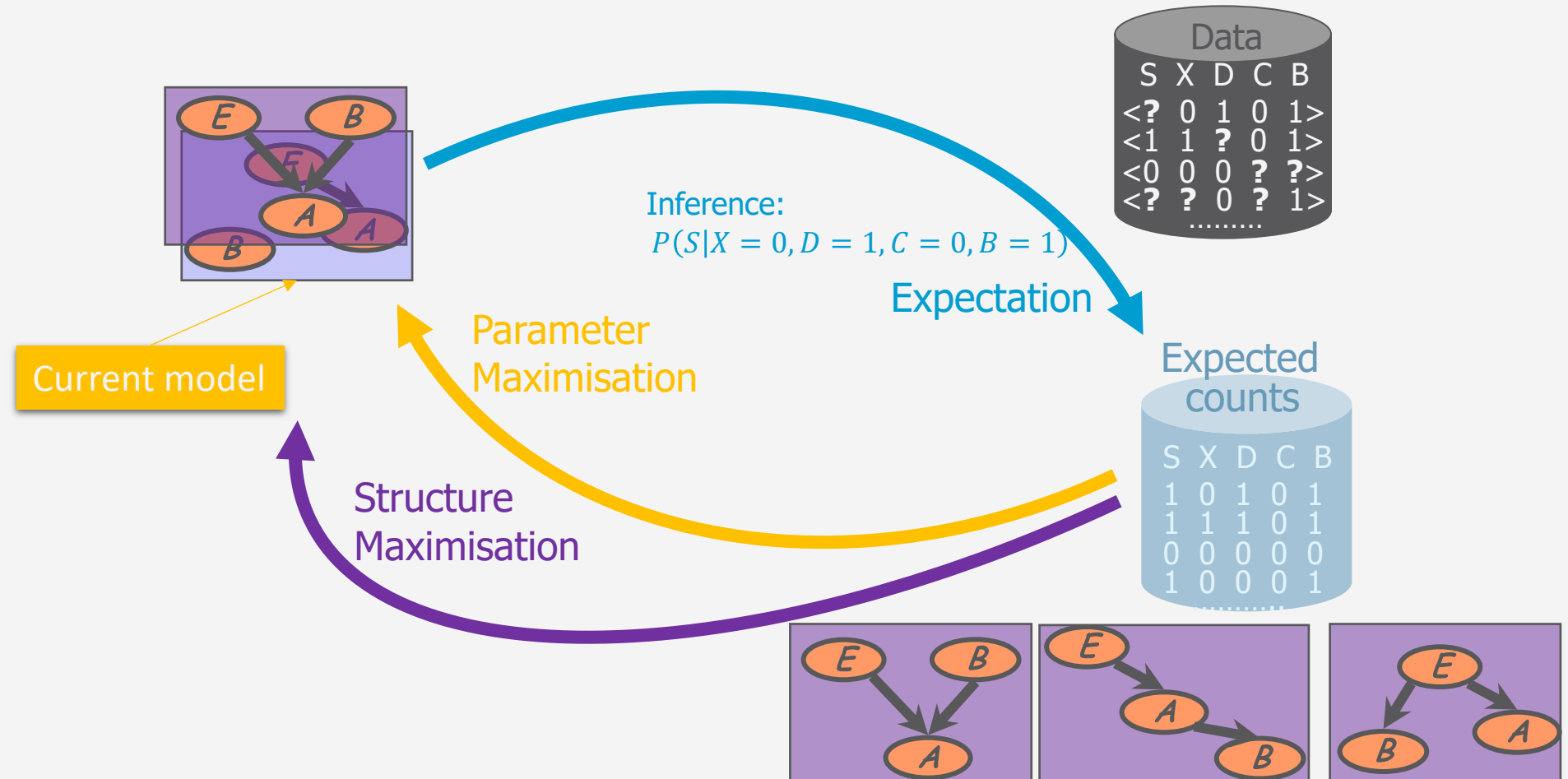
EM

- **EM**-Algorithm:
Iterate
until convergence



Structural EM

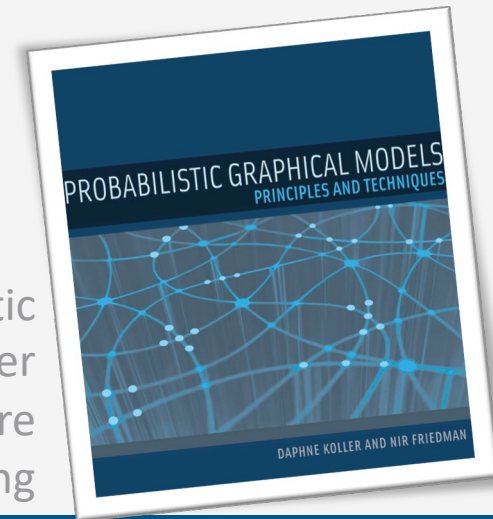
- **SEM**-Algorithm:
Iterate
until convergence



Bits and Pieces

- Can use approximate inference to compute **queries** asked during learning
- Can use alternative objectives such as
 - Pseudo-likelihood
 - Contrastive divergence
- Different approach to structure learning with fully observable data: Independence tests
 - Start with a fully connected graph, learn P_R MLE-style
 - Find which random variables are independent of each other, delete edges accordingly
 - Alternative: hypothesis tests to not learn P_R first but find independences for a factorisation; then, learn parameters MLE-style

See Ch. 20, in “Probabilistic Graphical Models” by Koller & Friedman (2009) for more details on learning



Interim Summary

- **Known structure, fully observable:**
only need to do parameter estimation
 - MLE: relative frequencies in BNs; in undirected models: need to handle $Z \neq 1$, when computing relative frequencies
- **Known structure, hidden variables:**
use expectation maximisation (EM) to estimate parameters
 - Cycle through computing expected counts
- **Unknown structure, fully observable:**
do heuristic search through structure space, then parameter estimation
- **Unknown structure, hidden variables:**
structural EM

Outline: 4. Learning

a. Relation Learning in Logic

- First-order inductive learning, first-order logical decision trees

b. Learning of PGMs

- Parameter and structure learning

*c. **Lifting PGMs to PRMs***

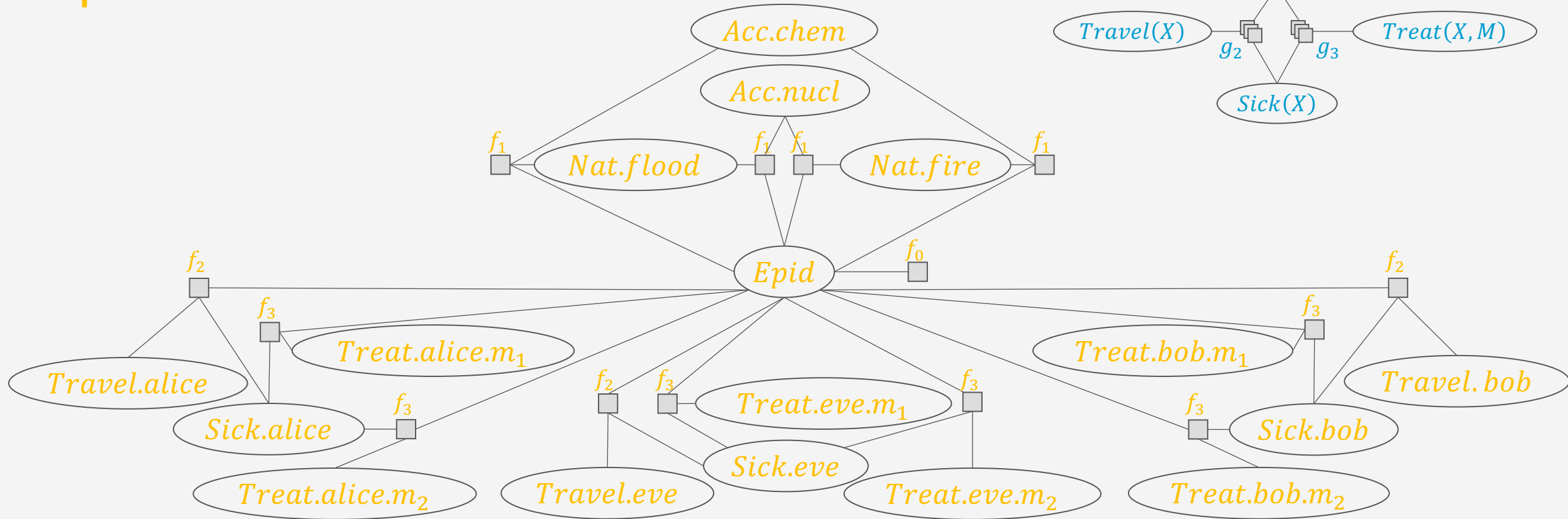
- Colour passing

d. PRM Learning

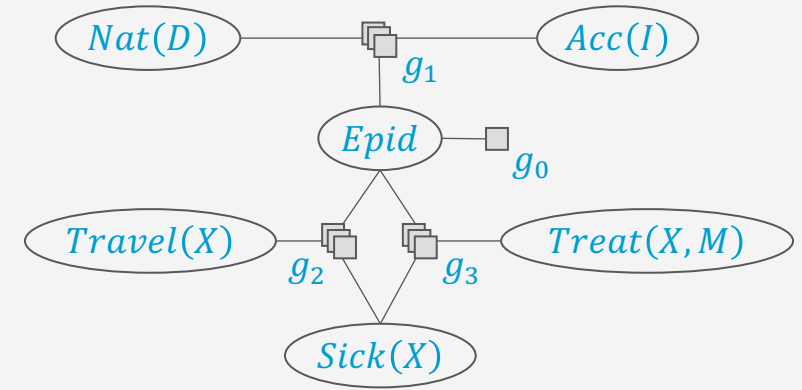
- Relational dependency network learning
- Changing domains

Compression

Input



Goal

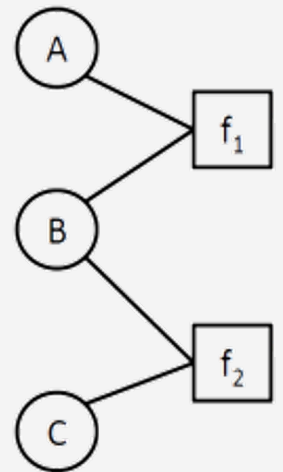


From Propositional Models to Lifted Models

- Having learned a propositional model, lift the model to a first-order representation
 - Identify symmetries in model
 - Subgraph isomorphisms
 - Exchangeabl function specifications
- **Colour passing algorithm**
 - Based on *Weisfeiler-Leman* subgraph isomorphism test, extended to include information on factors
 - Follows the idea of (loopy) belief propagation
 - Compress graph whenever nodes would send identical messages
 - Make symbolic message passes, sending colours instead of computed messages
 - Less computational effort
 - Use compressed graph for probabilistic inference

Compression: Pass the Colours Around

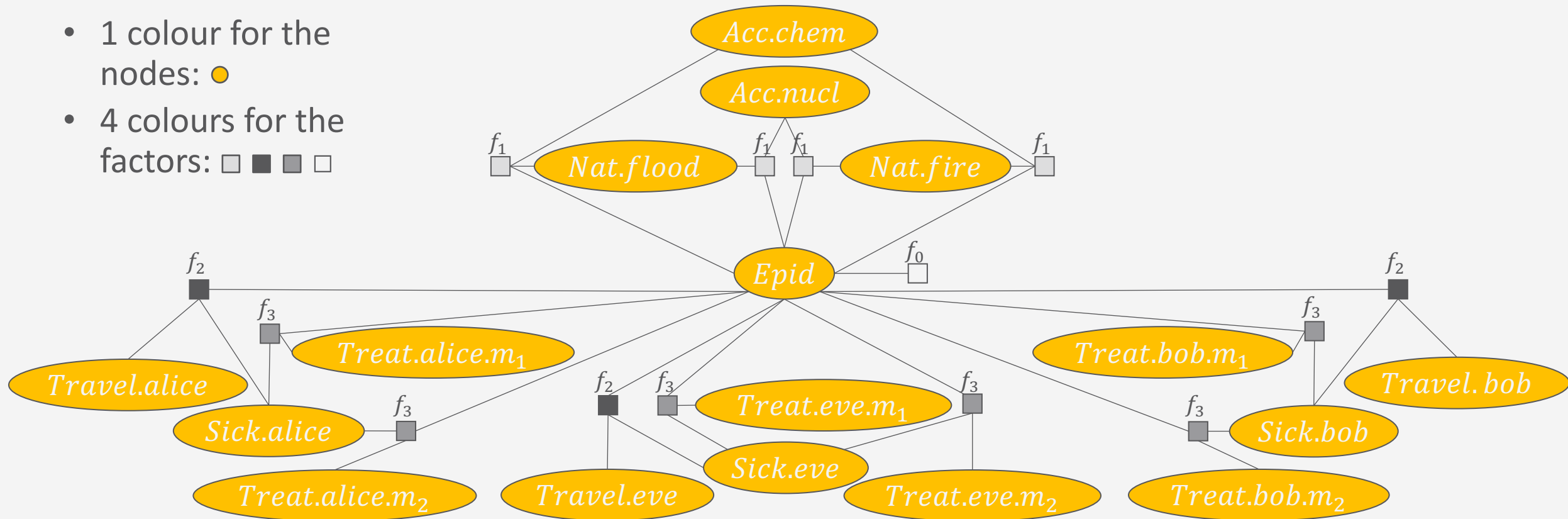
- Colour nodes according to the evidence you have
 - No evidence, say **red**
 - State “one”, say **brown**
 - State “two”, say **orange**
 - ...
- Colour factors distinctively according to their equivalences
 - For instance, assuming f_1 and f_2 to be identical and B appears at the second position within both, say **blue**



Compression

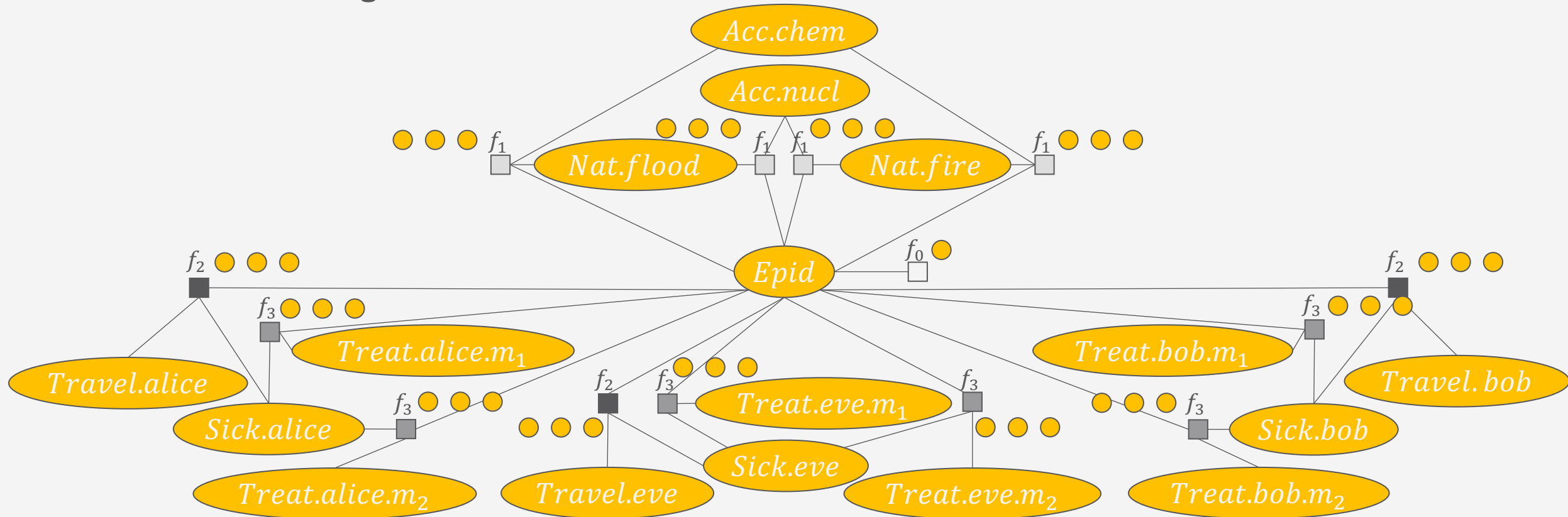
- Initialisation: Colour nodes and factors

- 1 colour for the nodes: ●
- 4 colours for the factors: □ □ □ □



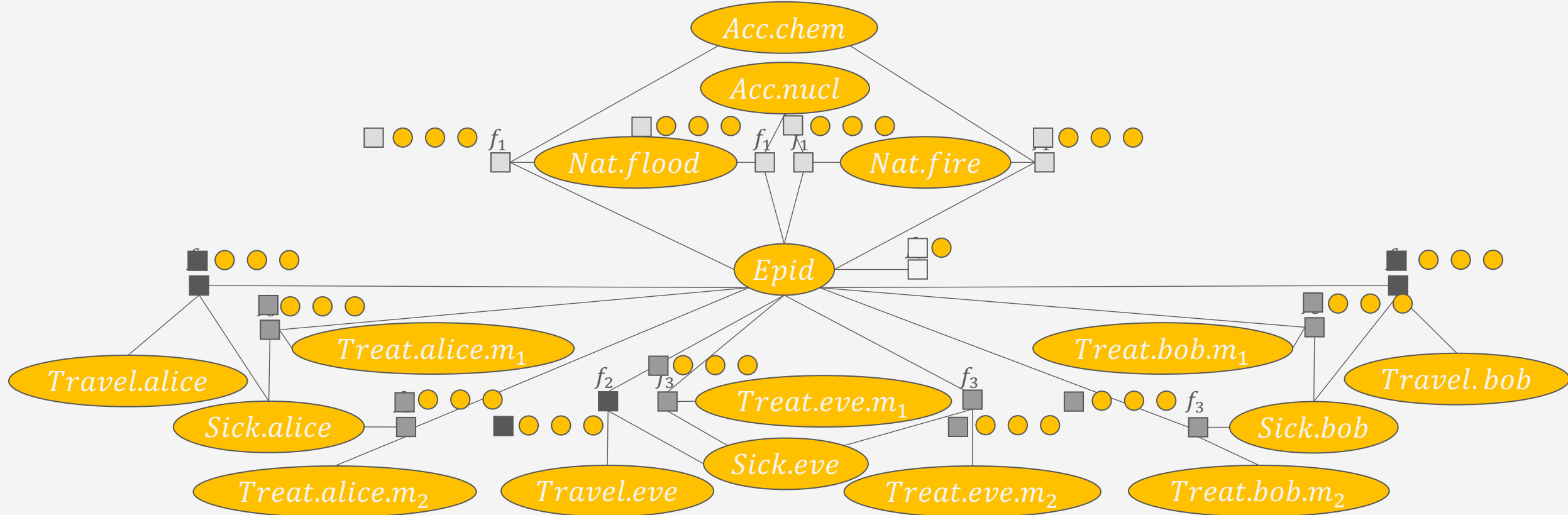
Compression

1. Factors collecting colours from nodes



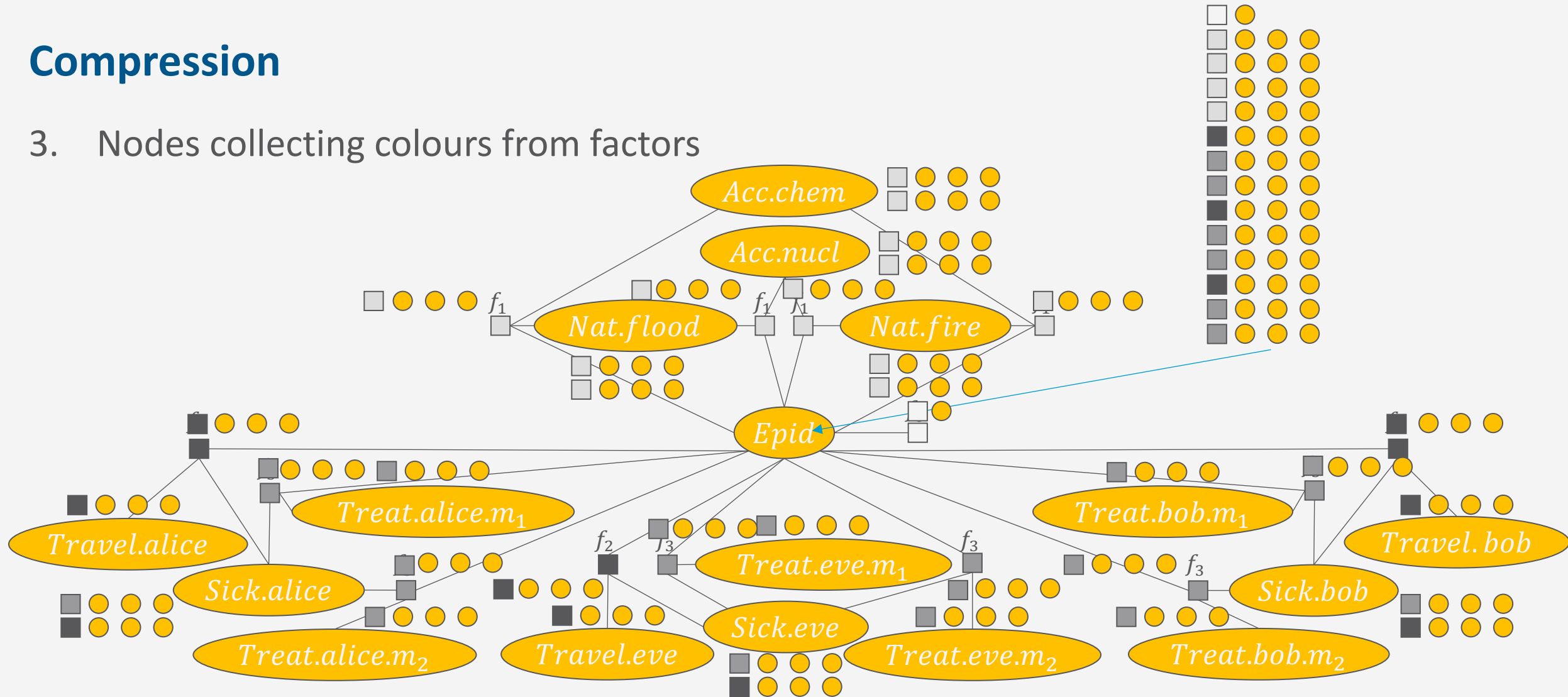
Compression

2. Factors signing their own colours to the collected ones



Compression

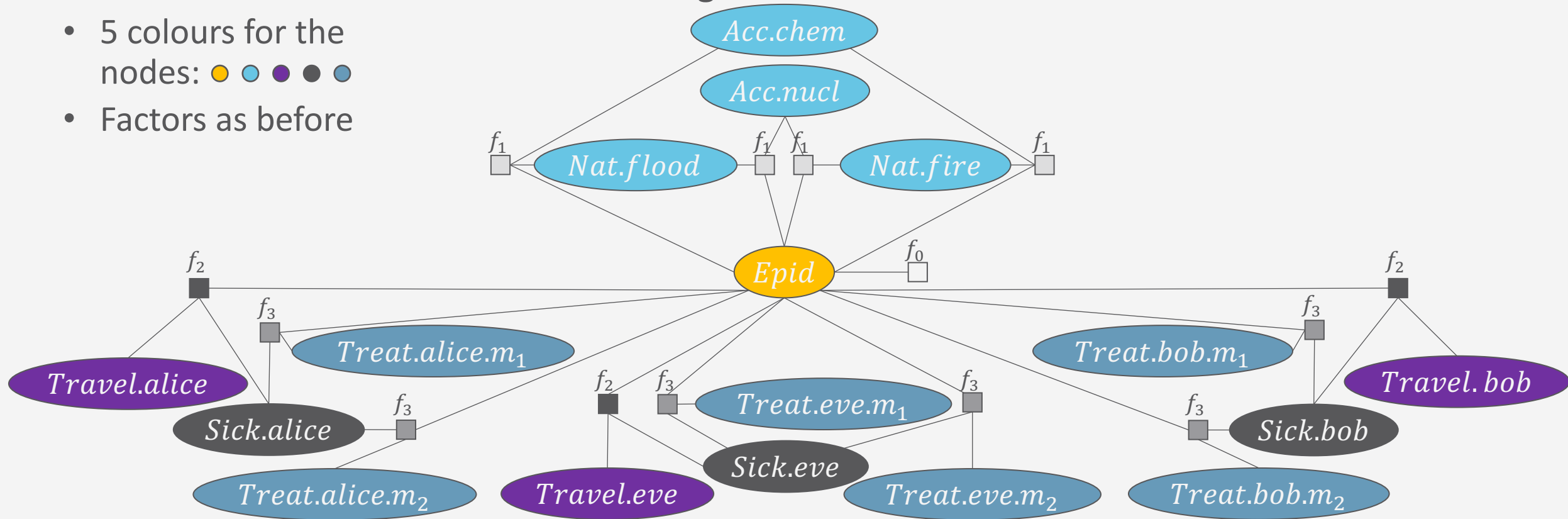
3. Nodes collecting colours from factors



Compression

4. Recolour nodes based on collected signatures

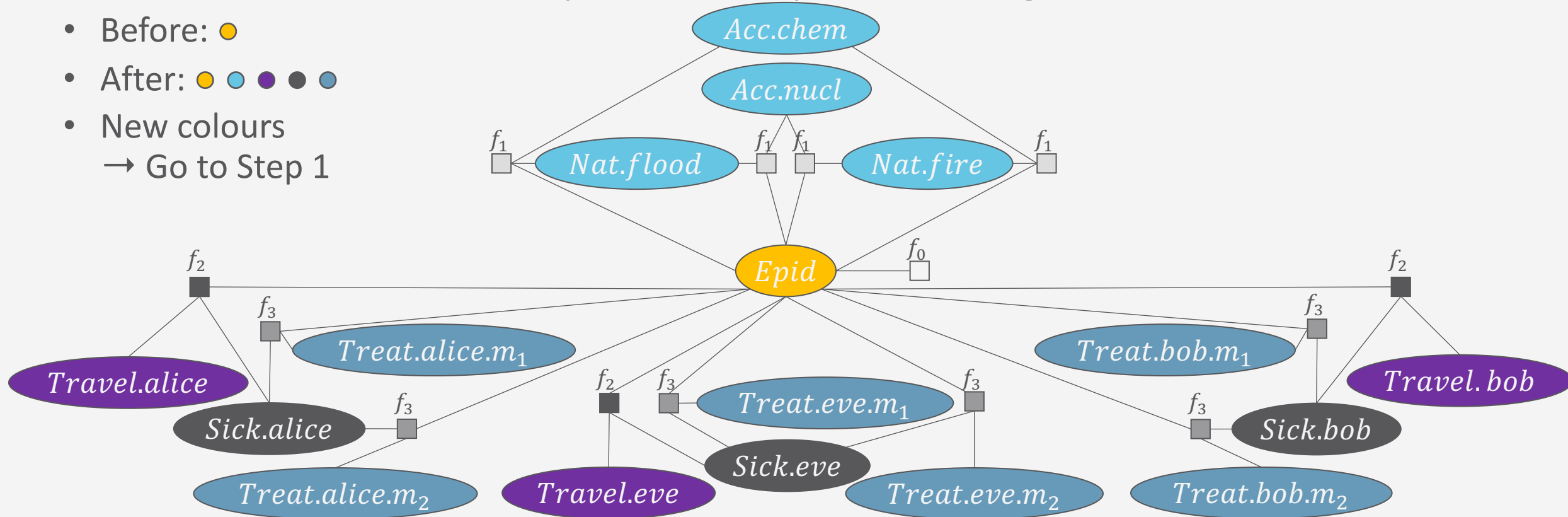
- 5 colours for the nodes: ● ● ● ● ●
- Factors as before



Compression

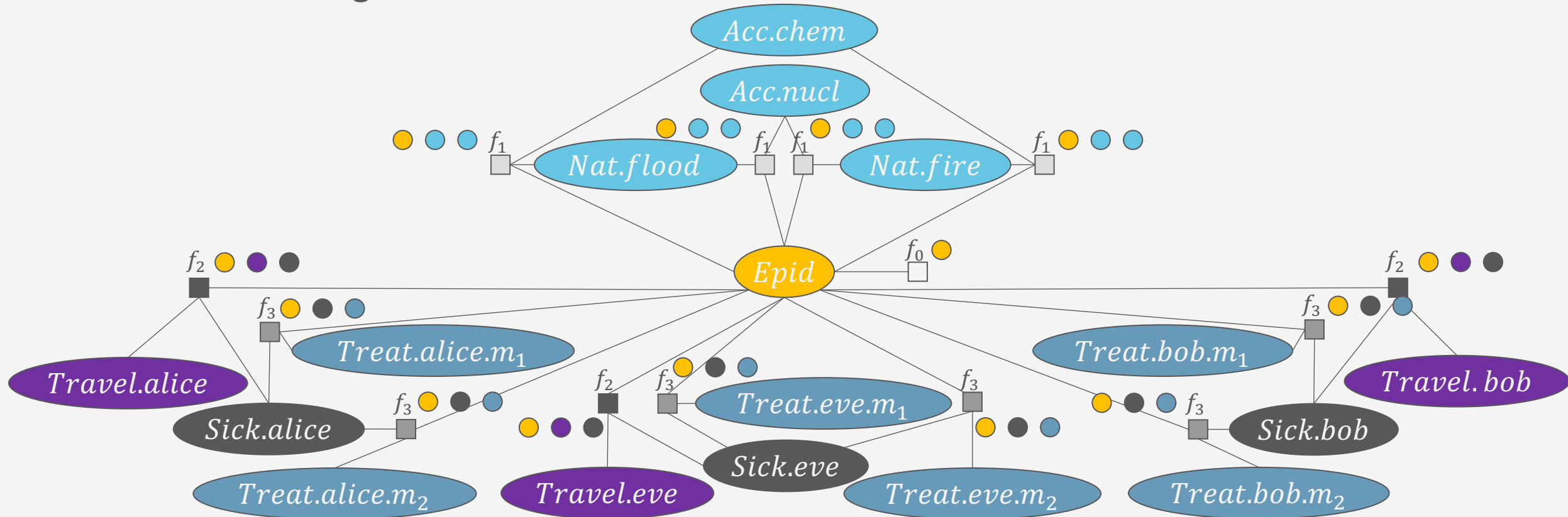
5. If no new colour created, stop. Otherwise, pass colours again.

- Before: ●
- After: ● ● ● ● ●
- New colours
→ Go to Step 1



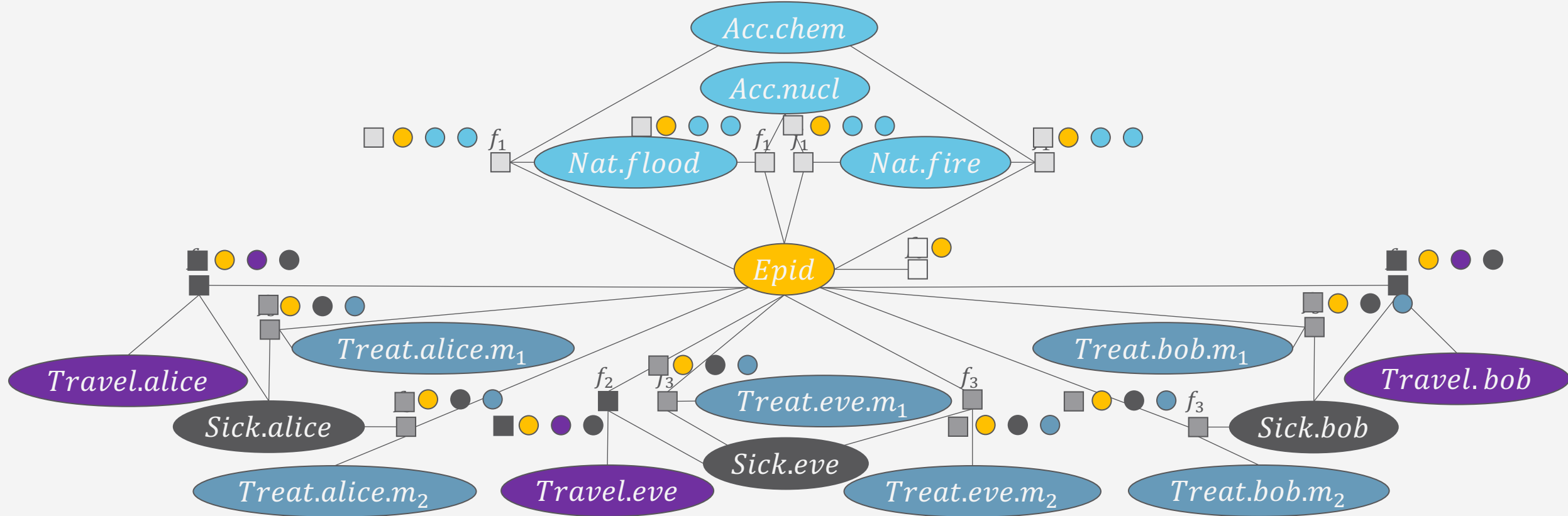
Compression

1. Factors collecting colours from nodes



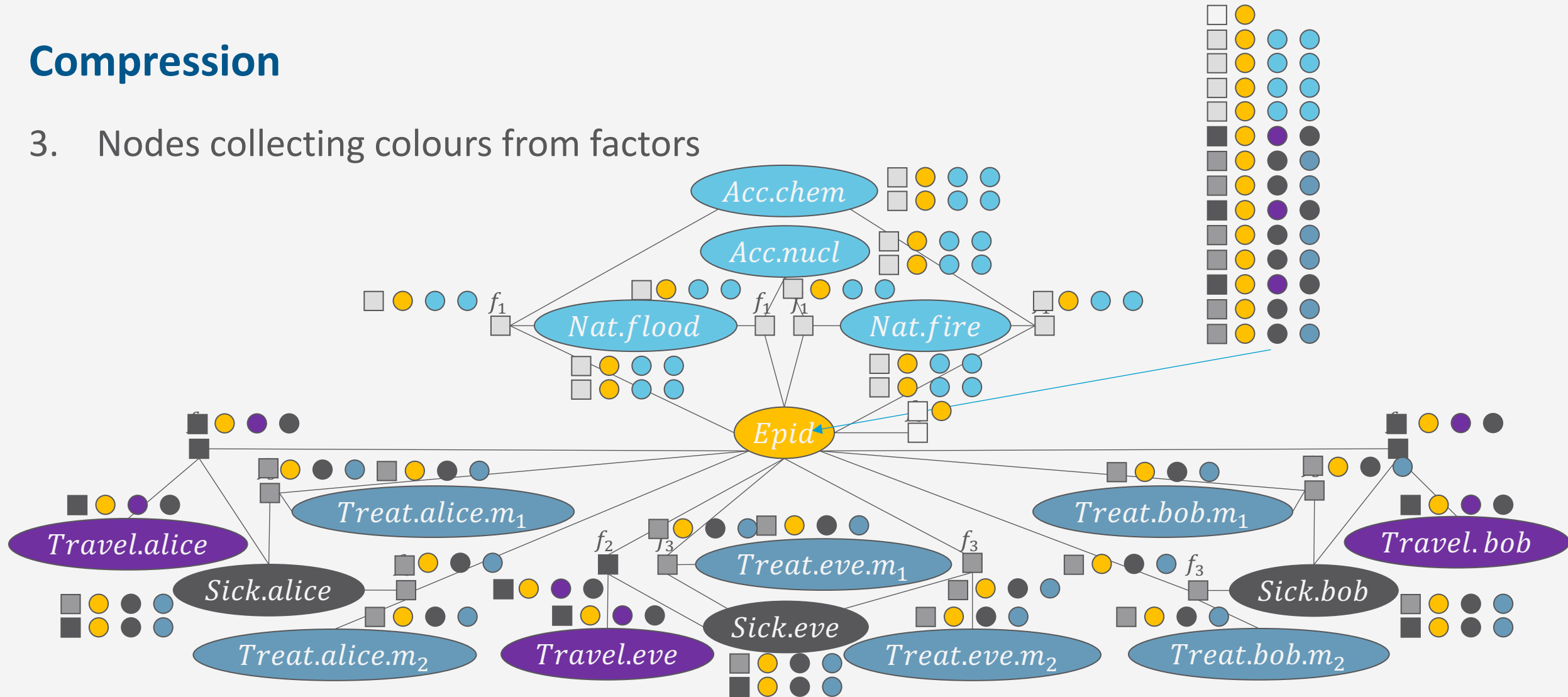
Compression

2. Factors signing their own colours to the collected ones



Compression

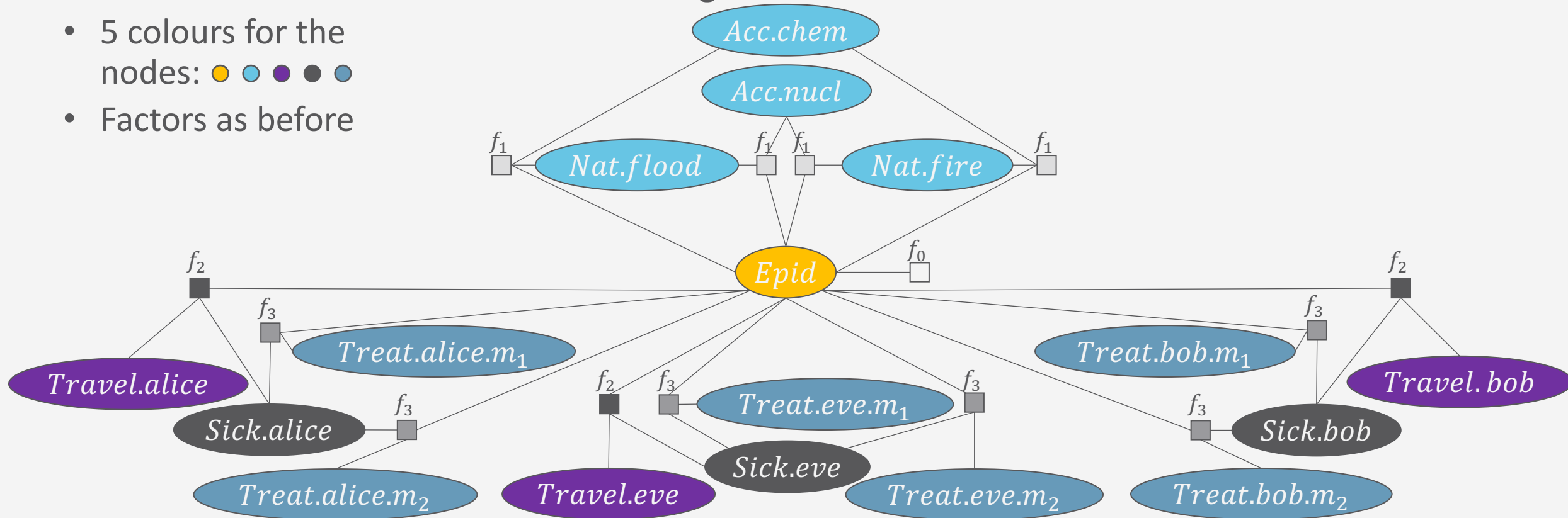
3. Nodes collecting colours from factors



Compression

4. Recolour nodes based on collected signatures

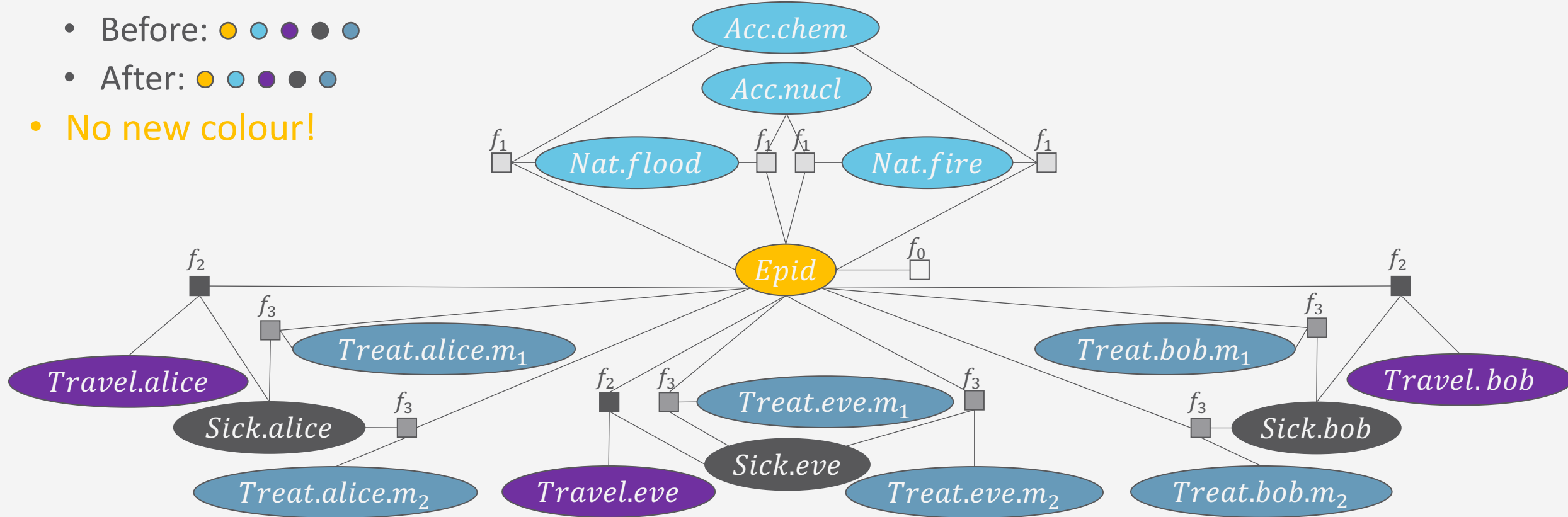
- 5 colours for the nodes: ● ● ● ● ●
- Factors as before



Compression

5. If no new colour created, stop. Otherwise, pass colours again.

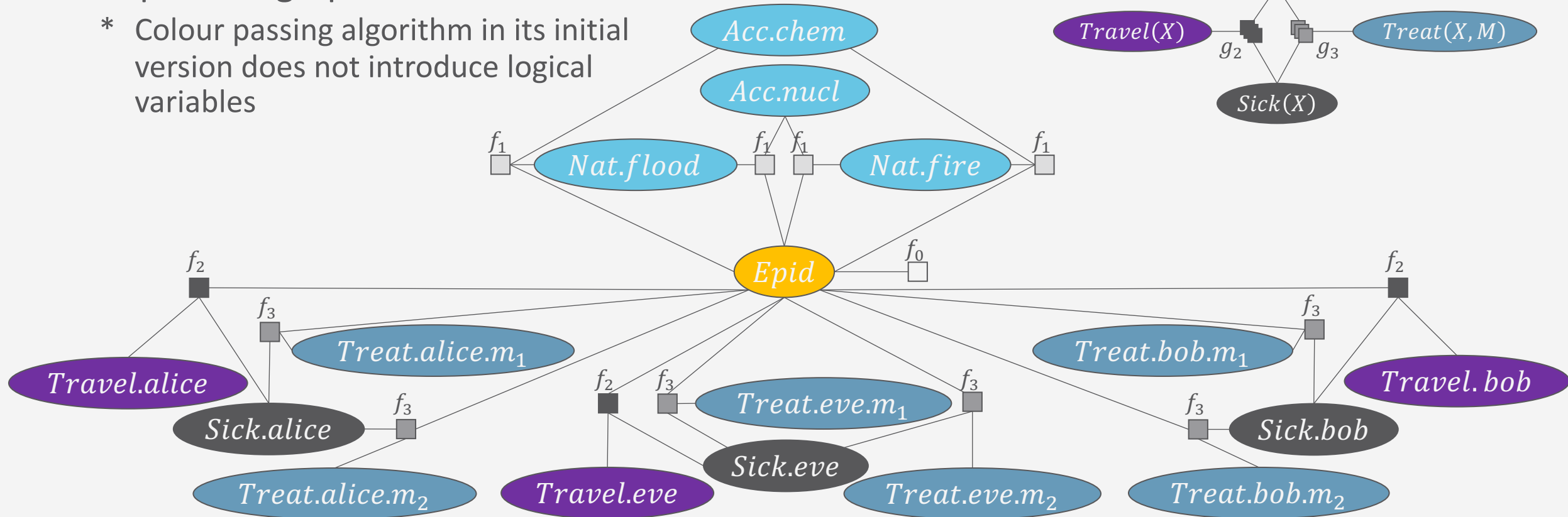
- Before: ● ● ● ● ●
- After: ● ● ● ● ●
- No new colour!



Compression

- Compressed graph*

* Colour passing algorithm in its initial version does not introduce logical variables

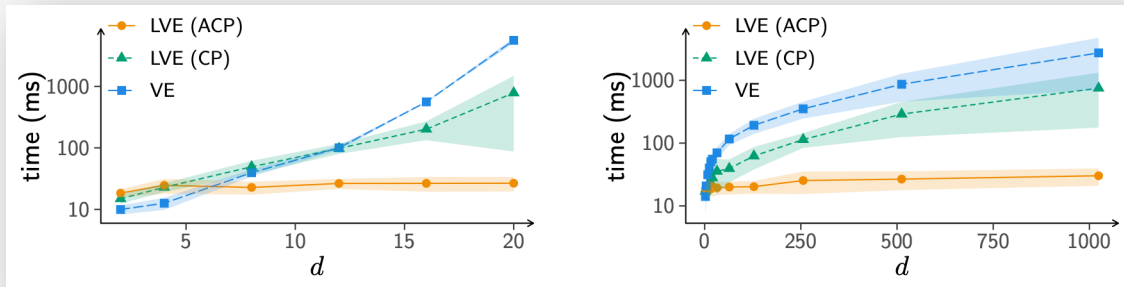


Colour Passing Compression

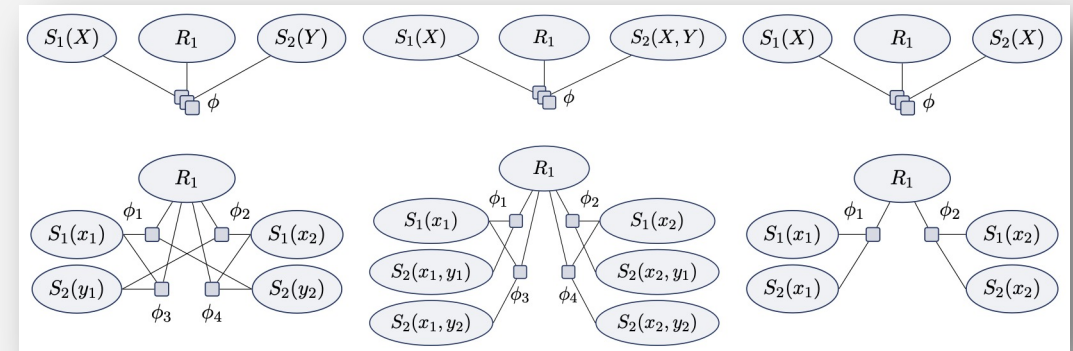
- Algorithm:
 - Initialisation: Colour each factor based on equality and each node according to evidence
 - 1. Each factor collects the colours of its neighbouring nodes
 - 2. Each factor “signs” its colour signature with its own colour
 - 3. Each node collects the signatures of its neighbouring factors
 - 4. Nodes are recoloured according to the collected signatures
 - 5. If no new colour is created stop, otherwise go back to 1
- Afterwards, build compressed version by combining random variables of same colour using logical variables
- Uses exact symmetries in factors
 - Same colour if factors considered equivalent (same potentials; arguments in fitting order)

Advanced Colour Passing (ACP)

- Check
 - if permuting inputs allows for aligning factors
 - for counting symmetries within a factor



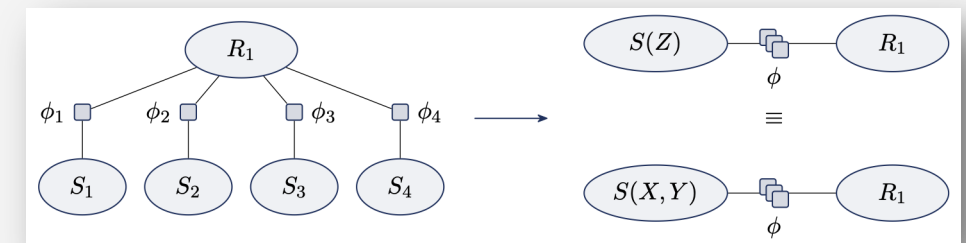
- Build a parfactor version
 - Three representative cases in the liftable two-variable fragment (see figure right above)
 - Can be ambiguous (see figure right below)



(a) Two log. vars.
non-overlapping

(b) Two log. vars.
with overlap

(c) One log. var.

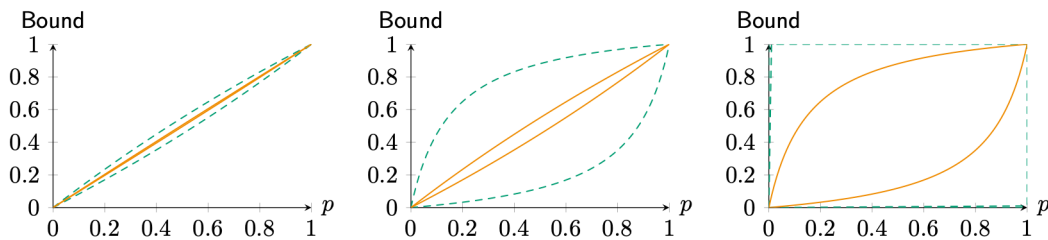


The four exchangeable factors can be represented using one log. var. with domain size 4 or two with size 2 each

Approximating Colour Passing

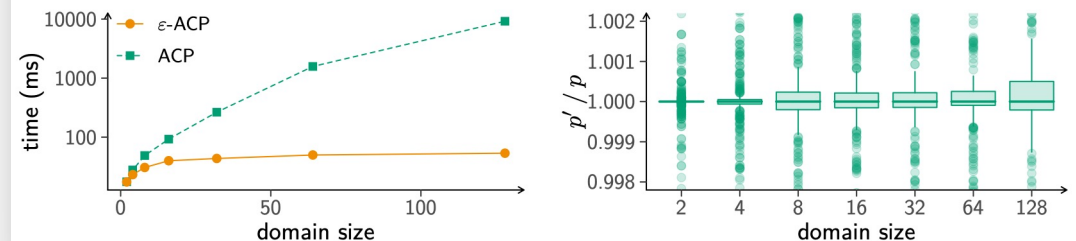
- ε -ACP: Approximate version allowing potentials to differ by a factor of $(1 \pm \varepsilon)$
 - E.g., consider $(1.0, 2.0)$ and $(1.1, 2.0)$ to be equivalent
 - Introduces ε as a hyperparameter
 - Possible to compute effect on query answers

- ▶ Bounds for $m = 10$ (left), $m = 100$ (middle), and $m = 1000$ (right) factors
- ▶ Dashed line: $\varepsilon = 0.01$, solid line: $\varepsilon = 0.001$
- ▶ x-axes depict the original probability p , y-axes reflect the bound on the change in p



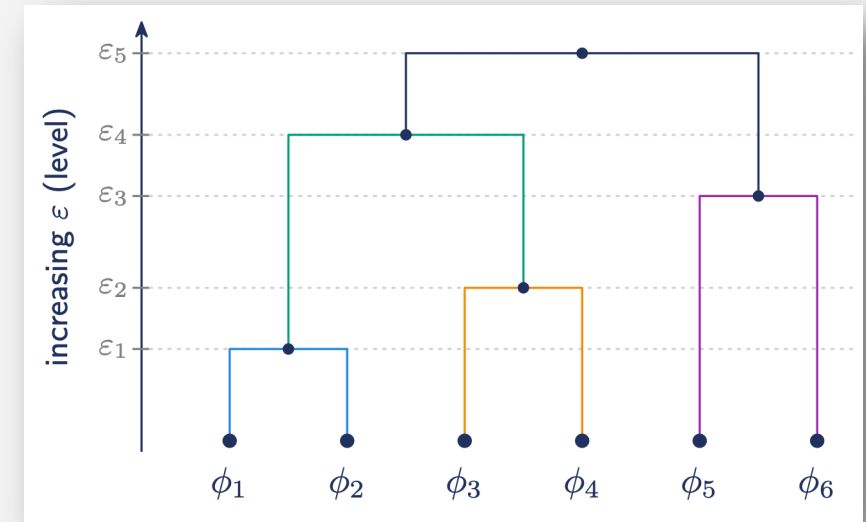
- ▶ Bounds apply to arbitrary queries and factor graphs

- ▶ Left: Comparison of run times for lifted inference
- ▶ Right: Quotients of query results p' in the modified factor graph and p in the original factor graph



Approximating Colour Passing

- **h-ACP**: Hierarchical version that builds increasingly compact models with increasing ε
 - Hyperparameter-free construction yielding a hierarchy of models
 - Structure preservation: Once grouped at some ε , factors stay grouped for every larger ε
 - Larger ε only merges smaller- ε groups
→ nested, interpretable models



Interim Summary

- Compress a model (lifted or ground) based on semantics
 - Pass colours around until convergence (no new colours)
 - Uses exact symmetries in factors
 - Same colour if factors considered equivalent
 - Ignores syntax
 - E.g., names of random variables
- “Literal” translation of propositional models into lifted models
 - Take the ground model, find the symmetries, combine them into a compact encoding
- Advanced colour passing to consider different argument orders, counting symmetries
- Approximate versions: ε -ACP, h-ACP

Outline: 4. Learning

a. Relation Learning in Logic

- First-order inductive learning, first-order logical decision trees

b. Learning of PGMs

- Parameter and structure learning

c. Lifting PGMs to PRMs

- Colour passing

d. PRM Learning

- Relational dependency network learning
- Changing domains

Relational Parameter Learning

- Assumption: individual instances in training data behave indistinguishably
 - Relational representation captures the setting with adequate accuracy
- Assuming relational structure is known
 - Complete data, e.g., using MLE
 - MLNs: decomposes per rule because of $\log \exp w = w$
 - PM: e.g., use IPF
 - Can use lifted inference for queries during learning
 - Data on groundings mapped to PRVs/predicates
 - Incomplete data: EM version
 - Could cluster instances into different domains and shatter model to increase accuracy
 - Trade-off between compact representation (no clustering) and accuracy (each instance in own cluster)

Structure Learning

- Can follow the same idea of structural EM
 - Already NP-hard problem in propositional setting
 - More complicated because there are not only random variables but also logical variables that can be combined together
- Other approaches
 - Learning approximate models, e.g.,
 - Relational dependency networks ([RDNs](#))
 - Using a relational probability tree for local distributions in RDNs
 - Boosted learning: Learn a set of distributions to approximate a local distribution ([RDN-Boost](#))

Learn Approximate Models

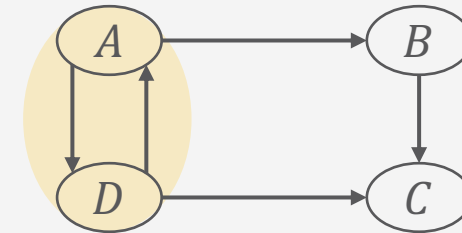
- Relational dependency networks (RDNs)
 - Using RPTs for local distributions in RDNs
 - Boosted learning: Learn a set of distributions to approximate a local distribution
 - Set of RRTs for local distributions in RDNs
 - Based on approximate propositional model of dependency networks (DN)
- Next slides
 - DNs
 - RDNs
 - Learning RPTs for RDNs
 - Boosted learning for RDNs

Dependency Networks

- Dependency network
 - Like a BN, i.e., a directed graph, but allowing for cycles
 - Each node corresponding to random variable R_k has a conditional probability distribution (CPD) $P(R_k | \text{parents}(R_k))$ assigned
- Approximate model
 - Represent joint distribution as a **product of (conditional) marginals**
 - Does not necessarily result in coherent joint distribution
 - If no cycles: exact (and equivalent to BN)
 - If discrete random variables and positive local CPDs
 - full joint recoverable [see Heckerman et al. (2000) for proof/details]
 - Allows for learning each distribution independently from the rest
 - Can work well with large amounts of data

→ Due to representing conditionals, better suited for classification

$$P_F(\mathbf{r}_f) = \frac{\prod_{f \in F} P(\mathbf{r}_f)}{\prod_{\{i,j\} \in E} P(\mathbf{r}_{ij})}$$



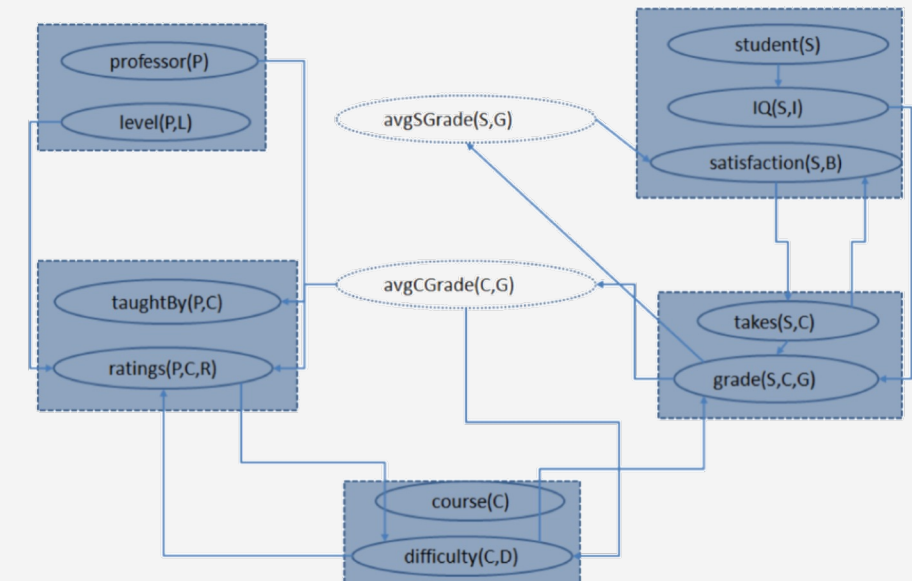
CPDs:

$P(A|D)$
 $P(B|A)$
 $P(C|B, D)$
 $P(D|A)$

Relational Dependency Networks

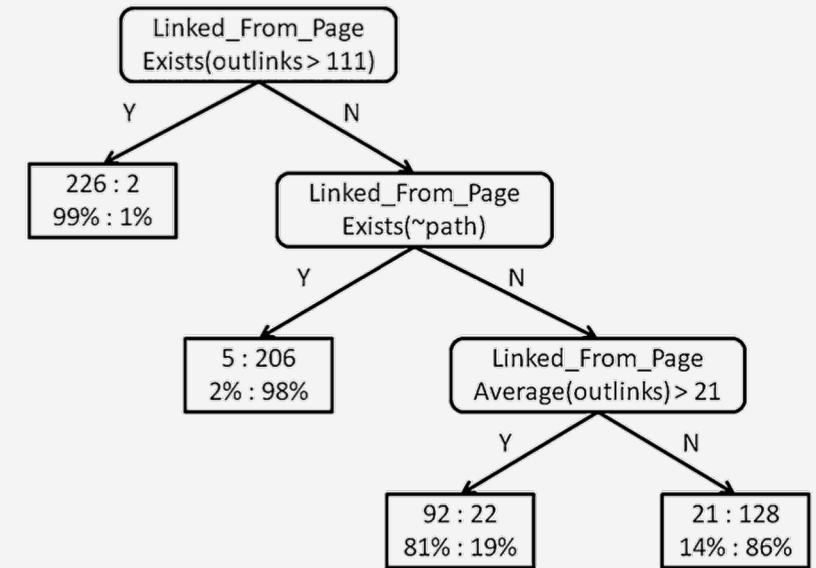
- Relational aspects explicitly modelled in DN
 - Relational databases as original motivation and backend for algorithms; logic perspective here
- Represent joint distribution as a **product of (conditional) marginals over ground atoms**
 - Inference by grounding and unrolling the model such that we have a BN again and then sampling on the ground BN
 - Unrolling the cycles in the model
- Each predicate A_k associated with a CPD $P(A_k | \text{parents}(A_k))$
 - **Aggregators** such as count, max, average

$$P_F(\mathbf{r}_f) = \frac{\prod_{f \in F} P(\mathbf{r}_f)}{\prod_{\{i,j\} \in E} \cancel{P(\mathbf{r}_{ij})}}$$



Learning RPTs for RDNs

- Represent CPD not as a table but as an RPT
- Learn RPTs individually for each (target) predicate
 - Construct aggregators: *mode, count, proportion, degree*
 - Inner nodes: decisions on aggregators
 - Actually restricted to aggregated predicates
 - Method: Recursive greedy partitioning
 - Split on feature that maximises the correlation between feature and class using χ^2 statistics
 - Pre-pruning with
 - p -value cut-off at $\frac{0.05}{\#attr}$
 - Depth cut-off at 7
 - Class distribution in leaves



χ^2 statistics: Calculate a so-called p -value as roughly the normalised sum of squared deviations between observed and theoretical frequencies

- Used in hypothesis testing to test, e.g., if observed values follow a theoretical distribution; given α , often $\alpha = 0.05$, if $p < \alpha$, reject H_0

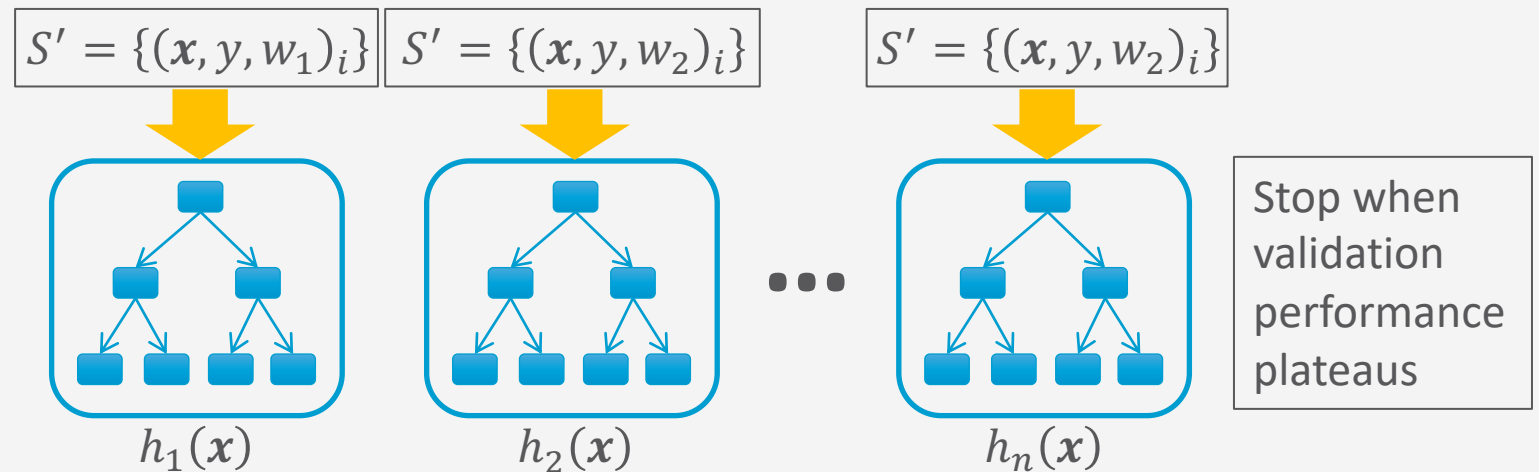
Boosting Idea

- *Ensemble* of (simple) classifiers ($\mathbf{x}$: feature vector, y : class labels)
 - Each classifier marginally better than random guessing
 - Idea: each classifier works well enough for a subset of the samples but not all of them
- Combine these weak classifiers to one strong classifier $h(\mathbf{x})$
 - Weighted sum:
 $h(\mathbf{x}) = \sum_i \alpha_i h_i(\mathbf{x})$
- E.g., *AdaBoost* with decision trees

Training loop:

- Train classifier $h_i(\mathbf{x})$ on current training set, add it to ensemble
- Find out which samples do not work well in ensemble, prioritise them, e.g., weight them higher (w_{i+1}), in the current training set S'

$$h(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \dots + \alpha_n h_n(\mathbf{x})$$



Functional-gradient Ascent

- Models given by $\frac{\exp \psi(y; \mathbf{x})}{\sum_{y'} \exp \psi(y'; \mathbf{x})}$
- Method for potential functions of models
 - Start with initial potential ψ_0 and iteratively add gradients Δ_i
 - After m iterations, potential is given by

$$\psi_m = \psi_0 + \Delta_1 + \dots + \Delta_m$$

- Δ_m is the functional gradient at iteration m and given by

$$\Delta_m = \eta_m \cdot E_{x,y} \left[\frac{\partial}{\partial \psi_{m-1}} \log P(y|\mathbf{x}; \psi_{m-1}) \right]$$

- η_m learning rate
- Basically, each Δ_m is a step in the direction of the gradient of the log likelihood function and η_m is the parameter that controls the step size

Functional Gradient Tree Boosting

- Since full joint unknown, treat data as surrogate
 - Instead of computing functional gradient over a potential function, functional gradients are computed for each training sample $y_i; \mathbf{x}_i$ (conditioned on potential from previous iteration)

$$\Delta_m(y_i; \mathbf{x}_i) = \nabla_{\psi} \sum_i \log P(y_i | \mathbf{x}_i; \psi) \Big|_{\psi_{m-1}}$$

- Set of $\Delta_m(y_i; \mathbf{x}_i)$ over all i form set of training examples
 - Train a function h_m that approximates Δ_m
 - Build/fit a regression tree h_m to minimise $\sum_i (h_m(y_i; \mathbf{x}_i) - \Delta_m(y_i; \mathbf{x}_i))^2$
 - Fitted function h_m not exactly the same as Δ_m but will point in same general direction (assuming enough training examples)
 - Then, the new potential at stage m is given by: $\psi_m = \psi_{m-1} + \eta_m h_m$
 - After M iterations, there are M regression trees to represent ψ

Functional Gradient for RDNs

- RDN represented as a set of conditional distributions $P(Y|\text{Pa}(Y))$ for all predicates Y
- Let $P(y|\text{Pa}(y))$ for a grounding y of Y be

$$P(y|\text{Pa}(y)) = \frac{\exp \psi(y; \mathbf{x})}{\sum_{y'} \exp \psi(y'; \mathbf{x})}$$

- $\psi(y; \mathbf{x})$ denotes potential function of y given all other $x, x \neq y$; y' iterates over groundings of Y
- Probability of example/grounding y_i of example i

$$P(y_i; \mathbf{x}_i) = \frac{\exp \psi(y_i; \mathbf{x}_i)}{\sum_{y'} \exp \psi(y'; \mathbf{x}_i)}$$

- Logarithm of $P(y_i; \mathbf{x}_i)$

$$\begin{aligned} \log P(y_i; \mathbf{x}_i) &= \frac{\log \exp \psi(y_i; \mathbf{x}_i)}{\log \sum_{y'} \exp \psi(y'; \mathbf{x}_i)} \\ &= \psi(y_i; \mathbf{x}_i) - \log \sum_{y'} \exp \psi(y'; \mathbf{x}_i) \end{aligned}$$

Functional Gradient for RDNs

$$\log P(y_i; \mathbf{x}_i) = \psi(y_i; \mathbf{x}_i) - \log \sum_{y'} \exp \psi(y'; \mathbf{x}_i)$$

- Functional gradient for y_i of example i

$$\begin{aligned} \Delta_m(y_i; \mathbf{x}_i) &= \frac{\partial \log P(y_i; \mathbf{x}_i)}{\partial \psi(y_i = 1; \mathbf{x}_i)} \\ &= \underbrace{I(y_i = 1; \mathbf{x}_i)}_{\text{Indicator function that returns}} - \frac{1}{\sum_{y'} \exp \psi(y'; \mathbf{x}_i)} \frac{\partial}{\partial \psi(y_i = 1; \mathbf{x}_i)} \sum_{y'} \exp \psi(y'; \mathbf{x}_i) \end{aligned}$$

Indicator function that returns

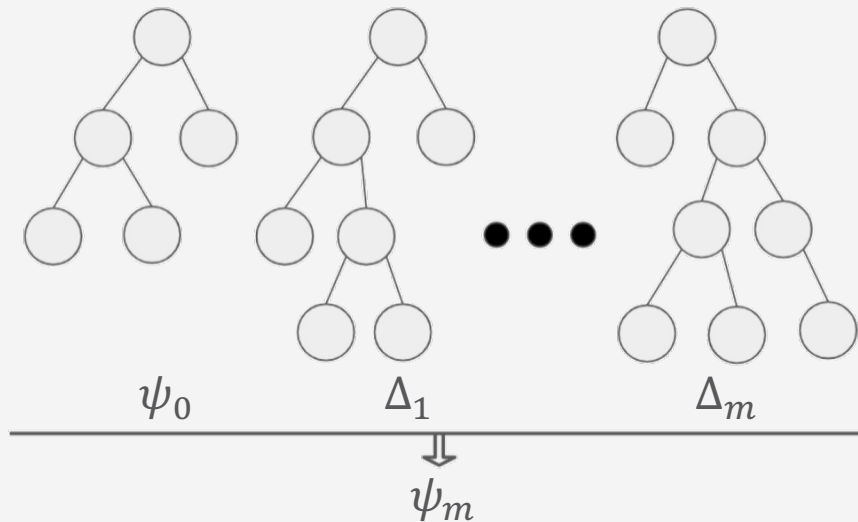
1 if $a_i = 1$ and 0 otherwise

$$= I(y_i = 1; \mathbf{x}_i) - \frac{\exp \psi(y_i = 1; \mathbf{x}_i)}{\sum_{y'} \exp \psi(y'; \mathbf{x}_i)} = I(y_i = 1; \mathbf{x}_i) - P(y_i = 1; \mathbf{x}_i)$$

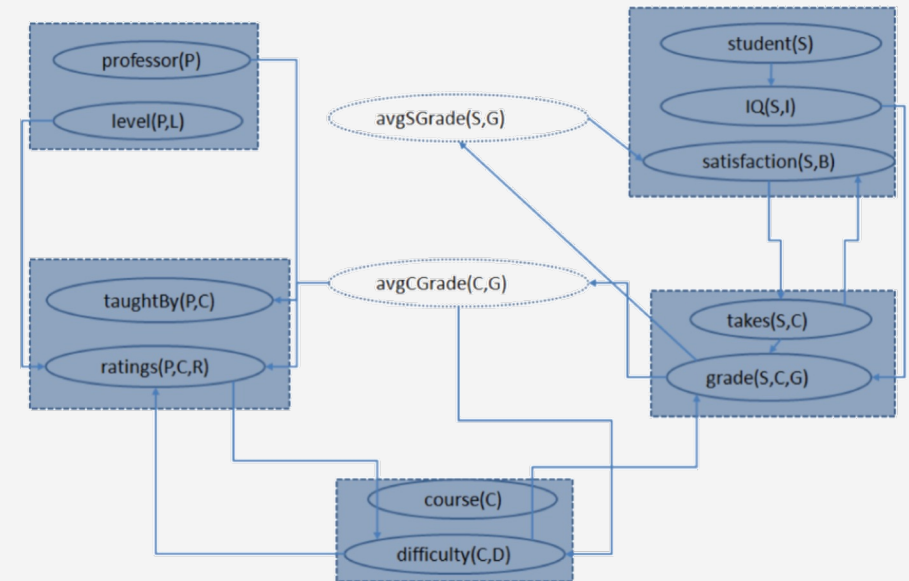
- Gradient at each example: adjustment required for the probabilities to match the observed value for that example
- Use $y_i; \mathbf{x}_i$ and $\Delta_m(y_i; \mathbf{x}_i)$ for all examples to fit an RRT

RDN-Boost: Overview

- For each $\psi(A_k; \text{parents}(A_k)) \in F$
 - I.e., for each predicate A_k
 - Build a set of RRTs, which form ψ
 - Each RRT estimates the gradient with which to update ψ
 - Using the gradient of each example of f as training examples



$$P_F(r_f) = \frac{\prod_{f \in F} P(r_f)}{\prod_{\{i,j\} \in E} P(r_{ij})}$$



Left figure: Siwen Yam, Devendra Singh Dhami, and Sriraam Natarajan: The Curious Case of Stacking Boosted Relational Dependency Networks. In 1st I Can't Believe It's Not Better Workshop (ICBINB@NeurIPS 2020), 2020.
Right figure: Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, Jude Shavlik: Gradient-based Boosting for Statistical Relational Learning: The Relational Dependency Network Case. In *Machine Learning*, 2012.

RDN-Boost

Function FitRelRegressTree returns an RRT fitted for S_k with a maximum of L leaves and a maximum depth D (cut-off criteria)

procedure TreeBoostForRDNs($Data$)

for $1 \leq k \leq K$ **do**

for $1 \leq m \leq M$ **do**

$S_k \leftarrow \text{GenExamples}(k; Data; F_{m-1}^k)$

$\Delta_m(k) \leftarrow \text{FitRelRegressTree}(S_k; L; D)$

$F_m^k \leftarrow F_{m-1}^k + \Delta_m(k)$

$P(a_k | \text{Pa}(A_k)) \propto \psi^k$

function GenExamples($k, Data, F$)

$S \leftarrow \emptyset$

for $1 \leq i \leq N_k$ **do**

 Compute $P(y_k^i = 1; x_k^i), x_k^i = \text{parents}(y_k^i)$

$\Delta(y_k^i; x_k^i) \leftarrow I(y_k^i = 1) - P(y_k^i = 1; x_k^i)$

$S \leftarrow S \cup \{(x_k^i, y_k^i), \Delta(y_k^i; x_k^i)\}$

return S

- Iterate through K predicates
- Iterate through M gradient steps
 - Generate examples
 - Functional gradient
 - Update model
- ψ^k is obtained by grounding F_M^k

- Iterative over all examples
- Probability of y_k^i being true
 - Compute gradient
- Update regression examples

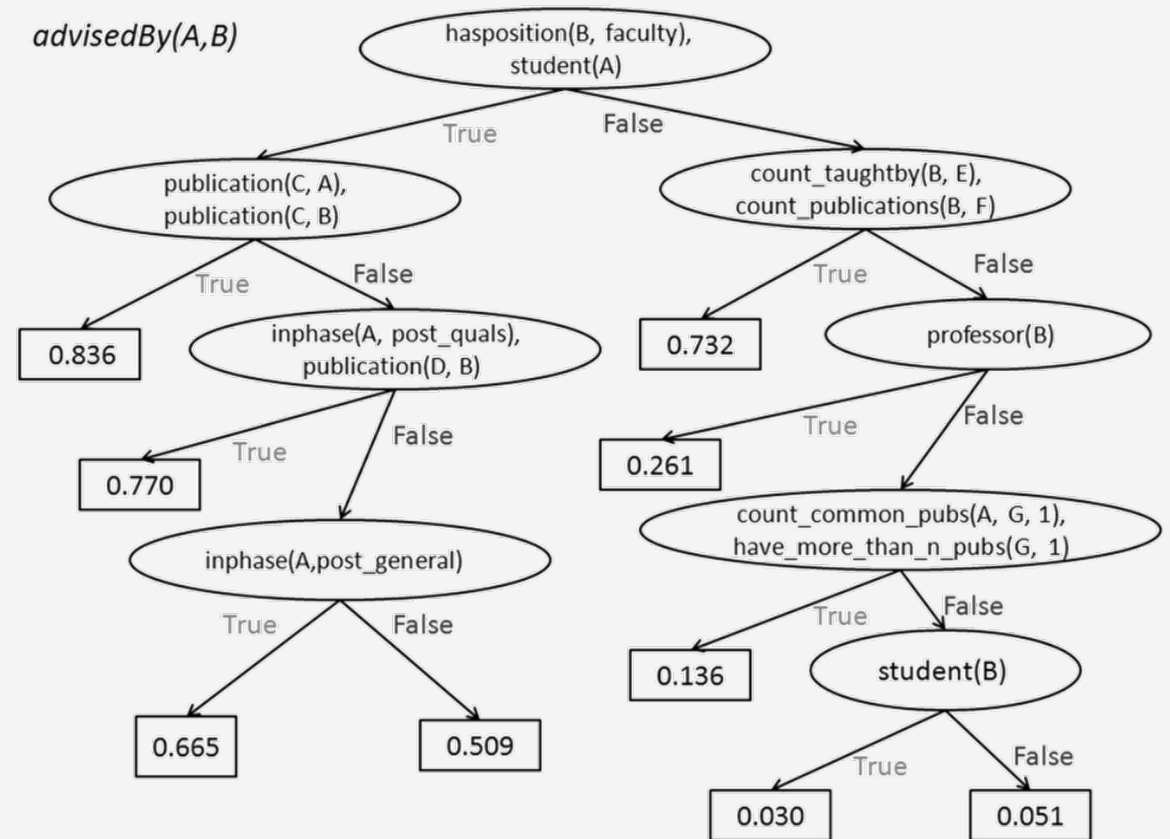
TreeBoostForRDNs

Bagging

- Ensemble method that reduces variance compared to boosting reducing bias
 - Compare: Random forests
 - Set of decision trees
 - Each tree learned on
 - Sampled subset of training instances
 - Sampled set of features available for each decision
- Combine both: Learn a set of boosted RDN models
 - Each run of RDN-Boost uses a sampled subset of the training examples
 - Only consider a random 50% of the candidate literals
- Increased prediction accuracy + easy parallelisation
 - Boosting does decrease variance with a large number of gradient steps, so bagging + boosting only has positive effect if considering small number of gradient steps

Interpretability

- Set of RRTs not really interpretable
 - Combine all RRTs into one tree and produce probabilities in leaves for interpretation by humans

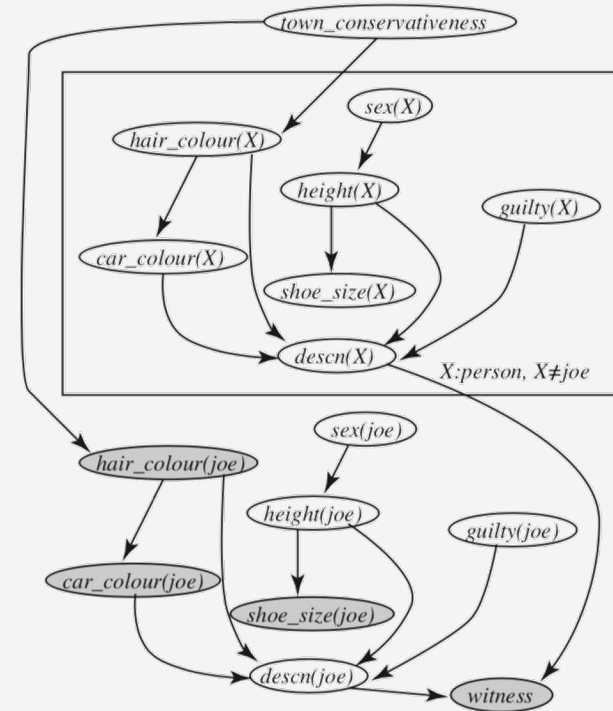


Interim Summary

- FOIL
 - Logic based: Learn relations given examples, background knowledge and a target concept
- First-order logical decision trees
 - Logic based: Learn relations in tree form given examples, background knowledge and a target concept
- Regression/probability trees
 - Leaves with continuous values/probability distributions
 - Relational versions: predicates in inner nodes
 - Represent conditional distributions as trees
 - Boosting
 - Set of trees to represent distributions
- Can construct weighted FOL formulas to build an MLN and then convert it to whatever form one needs (FOKC; LVE/LJT)
- Very little work on learning the structure of general PRMs

Changing Domains

What happens after we have learned a model and domains change?

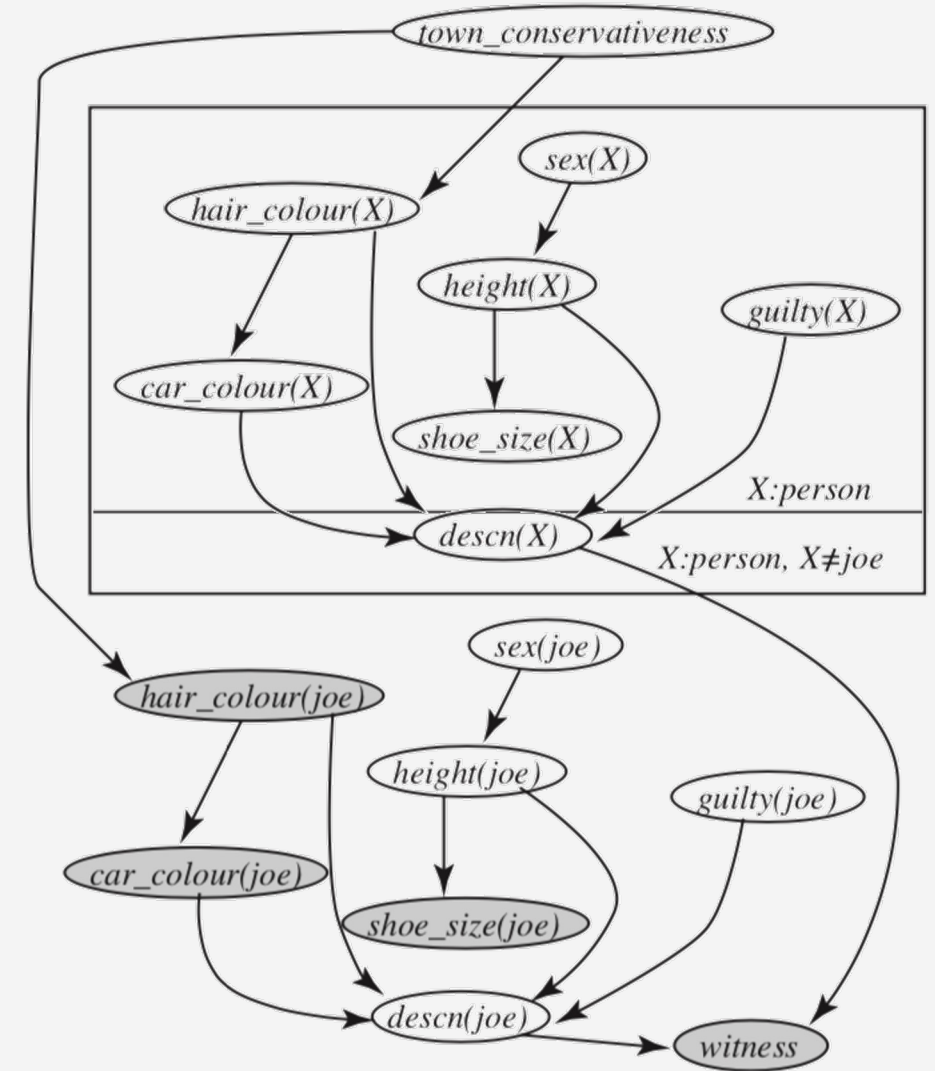
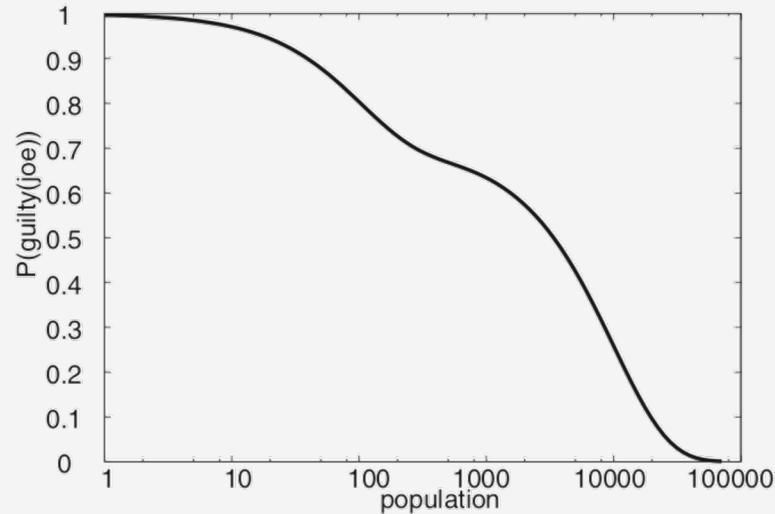


Known Universe

- Lifting based on knowing your universe
 - Universe = set of domains for logical variables in model
- Grounding semantics is only defined given specific domains for logical variables
 - Evidence for known constants
 - Queries reference known constants
- Potentials in parfactors learned with an underlying, specific universe size
 - “Make sense” for that size
- What if domains change?
 - Can we transfer the learned model to the new setting?
What needs to change? Does something need to change

Changing Domains

- Keep semantics as before
 - Assume that parafactors accurately describe world
- Posterior probabilities change depending on domain sizes
 - Example by Poole (2003)



... Without Effects

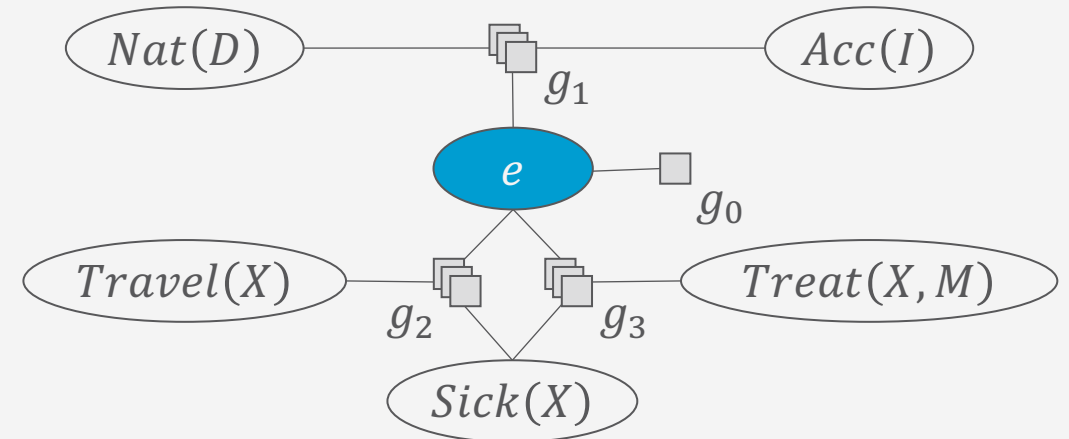
- (Conditional) Independence

PRVs, containing logical variables X , that are (conditionally) independent from query terms
 → domains of X have no influence on query results

- E.g., given $Epid = e$,

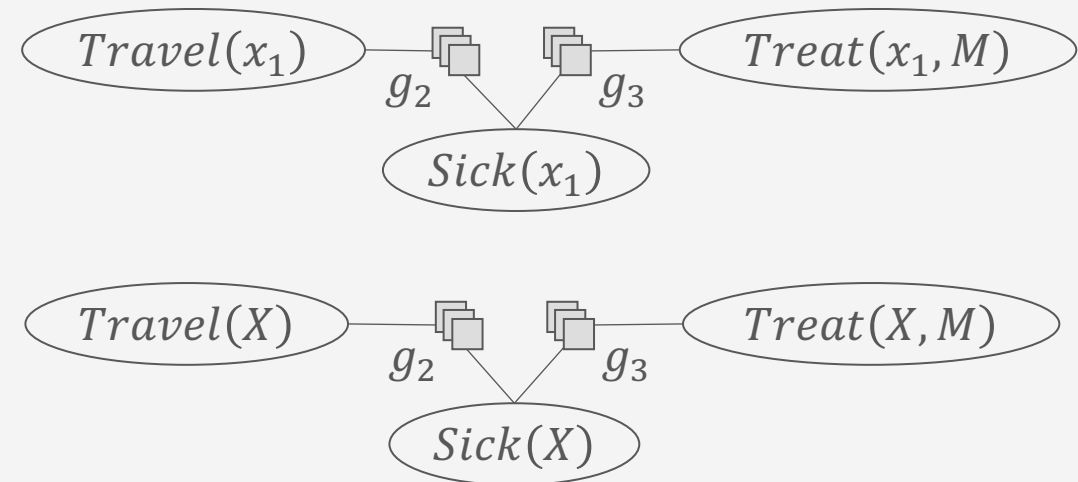
- $\text{dom}(D)$ and $\text{dom}(I)$ do not matter for queries regarding *Travel*, *Sick*, and *Treat*
- $\text{dom}(X)$ and $\text{dom}(M)$ do not matter for queries regarding *Nat* and *Man*

→ Partly invariant under increasing domain sizes



... Without Effects

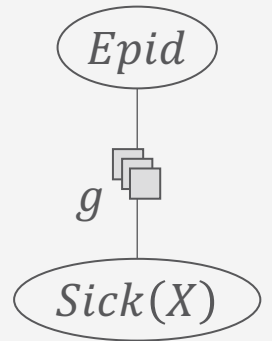
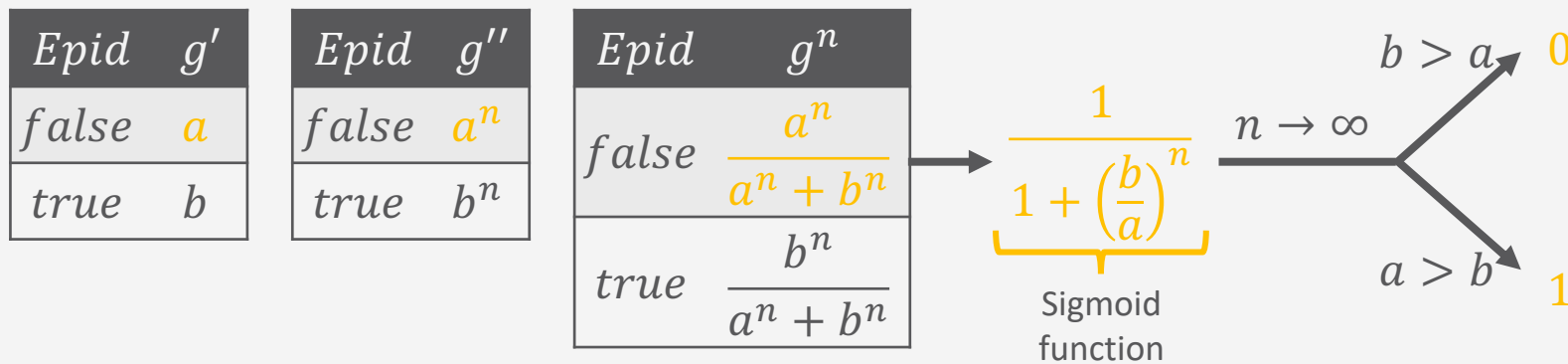
- A simple case of so-called **projectivity**
 - After shattering, query terms are independent of model parts containing logical variables X
 - domains of X have no influence on query results
 - Depends on model structure
 - More by Jaeger and Schulte (2018)
 - E.g., $P(\text{Sick}(x_1))$
 - $\text{dom}(X) = \{x_1, \dots, x_n\}$
 - After shattering:
 - $\text{dom}(X) = \{x_2, \dots, x_n\}$
 - Upper part independent from lower part; $\text{dom}(X)$ irrelevant
- Partly invariant under increasing domain sizes



Growing Domain Sizes

- Let domain size n grow
 - With grounding semantics, posteriors change
 - Can lead to **extreme** behaviour in the posteriors
- Example: *Epid* gets more and more neighbours with n rising

$$P(Epid) \propto \left(\sum_{s \in \text{ran}(Sick(X))} g(Epid, Sick(x) = s) \right)^n = (g'(Epid))^n = g''(Epid) = g^n(Epid)$$



Growing Domain Sizes

- How to avoid extreme behaviour?
 - Adapt values in model dependent on domain size
 - Approach for MLNs: **Domain-size aware MLNs**
 - Assume predicates $P_1, \dots, P_m$ occur in a FOL formula F
 - Count number of connections c_j for each predicate P_j given *new* domains
 - Build a connection vector $[c_1, \dots, c_m]$
 - Choose $\max_{c_i}[c_1, \dots, c_m]$ as scaling-down factor
 - Instead of max, other functions possible
 - Works best if the values in $[c_1, \dots, c_m]$ do not vary that much
 - Given an MLN with a set of formulas F_i with weights w_i
 - Rescale each w_i with scaling-down factor s_i computed for F_i as $\frac{w_i}{s_i}$
 - Analogous approach possible for parfactors

Outline: 4. Learning

a. Relation Learning in Logic

- First-order inductive learning, first-order logical decision trees

b. Learning of PGMs

- Parameter and structure learning

c. Lifting PGMs to PRMs

- Colour passing

d. PRM Learning

- Relational dependency network learning
- Changing domains

→ Next: Sequential Models and Inference