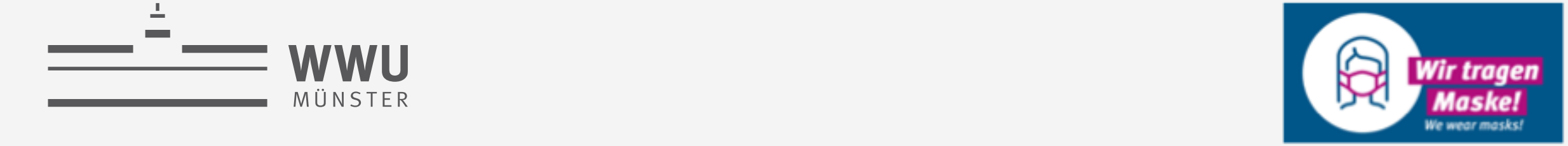
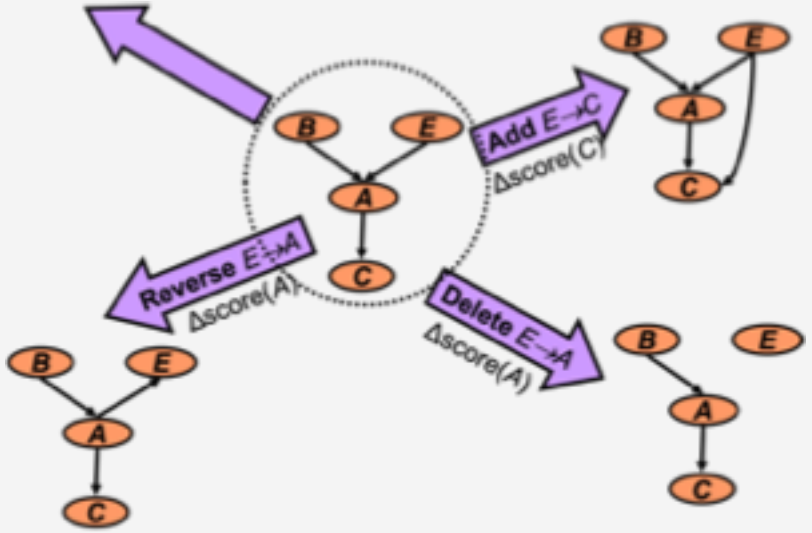
WWU
MÜNSTER



Lifted Learning

Statistical Relational Artificial Intelligence
(StaRAI)

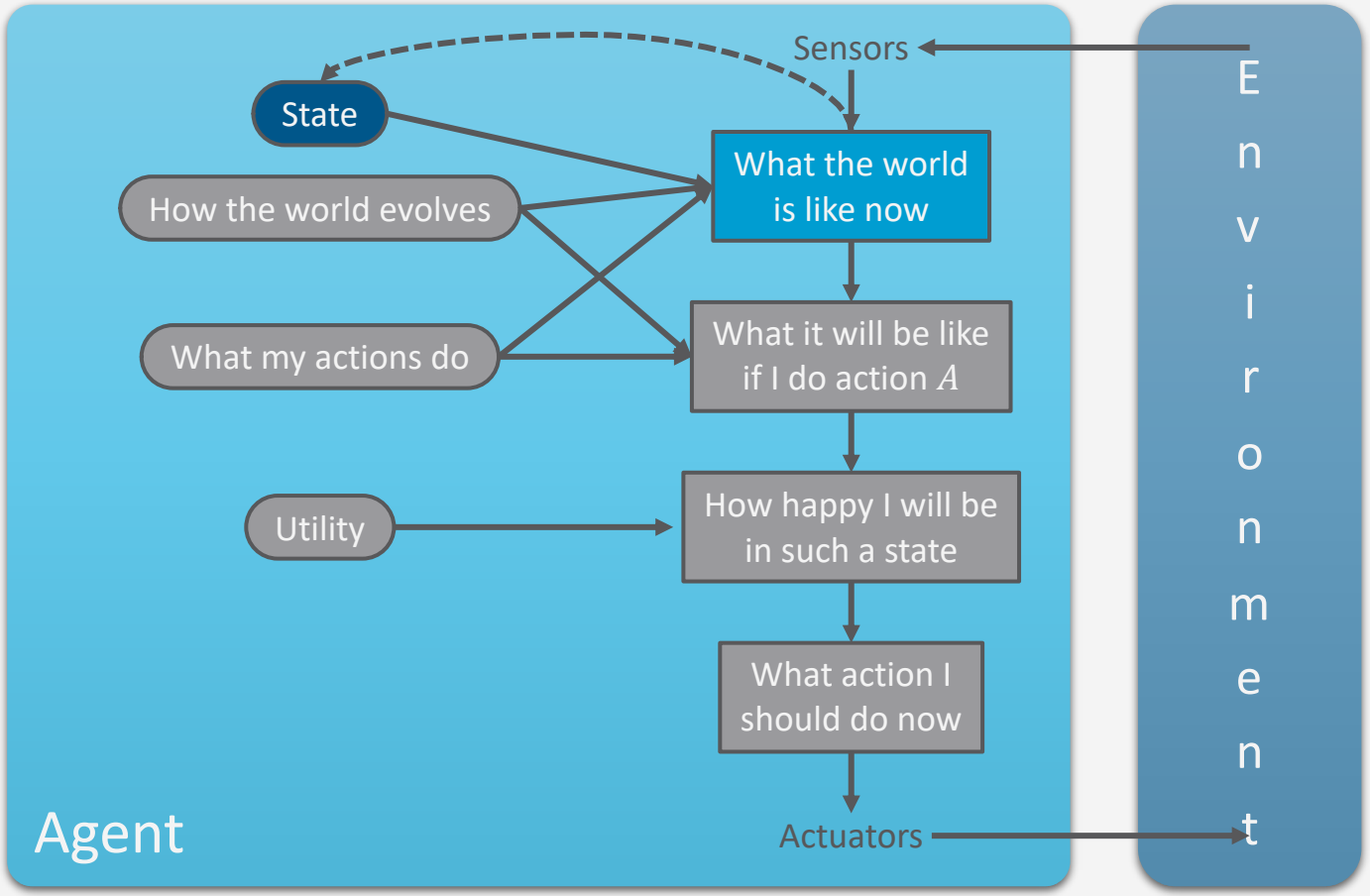


Tanya Braun

Contents

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. Lifted Learning

A. Overview of (propositional) learning
- Parameter and structure learning

B. Lifted encoding of propositional models
- Colour passing

C. Relation(al) learning
- First-order inductive learning, decision tree representation
- Relational dependency network learning
- Changing domains

D. Approximating Symmetries
- Evidence and model-based approaches, local structure

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 → 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 → query answering needs to consider all models and combine answers of all (weighted by likelihood of model)

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

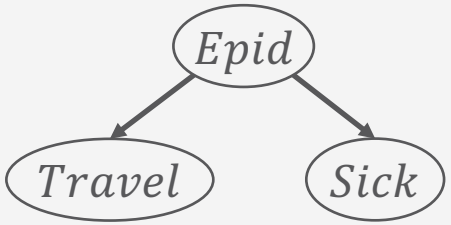
independent and identically distributed (iid) data points

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

General ML procedure

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

Model



Data

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

→ Probability distributions to learn

$P(Epid), P(Travel \mid Epid), P(Sick \mid Epid)$

$Epid$	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

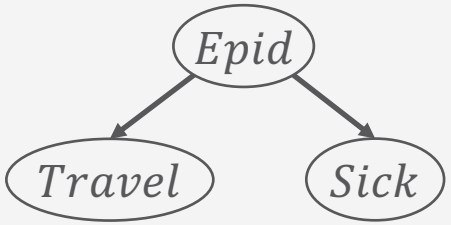
$Epid$	$Sick$	P
$false$	$false$	$1-\theta_3$
$false$	$true$	θ_3
$true$	$false$	$1-\theta_4$
$true$	$true$	θ_4

ML Parameter Estimation in BNs

- Estimating parameters in BNs with *known structure* and *data for all variables* simplest learning problem
- Goal: Estimate parameters $\boldsymbol{\theta}$
  - Entries in CPTs $P(R|\mathrm{Pa}(R))$

- ML procedure leads to relative frequencies, i.e., counting occurrences, e.g., for θ_0, θ_1:
  - $\theta_0 = \frac{\#(datapoints\ with\ Epid{=}true)}{\#(all\ datapoints)}$
  - $\theta_1 = \frac{\#(datapoints\ with\ Epid{=}false, Travel{=}true)}{\#(datapoints\ with\ Epid{=}false)}$

Model



Data

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

→ Probability distributions to learn

$P(Epid), P(Travel \mid Epid), P(Sick \mid Epid)$

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

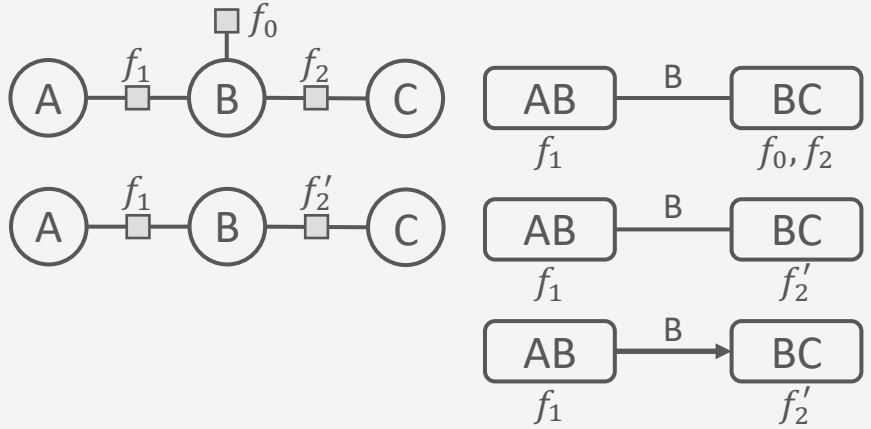
Epid	Sick	P
false	false	$1-\theta_3$
false	true	θ_3
true	false	$1-\theta_4$
true	true	θ_4

ML Parameter Estimation in Factor-based Models

- Problem: normalisation constant Z
  - $\prod_{j=1}^{N} P(d_j|\boldsymbol{\theta}) = \prod_{j=1}^{N} \frac{1}{Z} P(d_j, \boldsymbol{\theta})$
  - Combines all parameters in one expression
    - $Z = 1$ in BNs: Learning decomposes into learning each entry individually
- ML procedure only leads to statement that
  - ML estimates should be in such a way that the model marginals $P(\boldsymbol{r}_f)$ are equal to the normalised empirical counts:

$$\frac{\#(\boldsymbol{r}_f)}{N} = P(\boldsymbol{r}_f)$$

- Solutions?
  - Could enforce CPTs in factors, making $Z = 1$
    - Similarly to what we did for sampling in factor-based models: Transform model and learn parameters MLE-style



  - If keeping the factors: *Find local optimum using an iterative updating procedure*

Iterative Proportional Fitting (IPF) with Junction Trees

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

Compute $P^t(\boldsymbol{r}_f)$ using current m_{ji}^t and F_i^t
- Organise in a reasonable way over all local factors and assignments

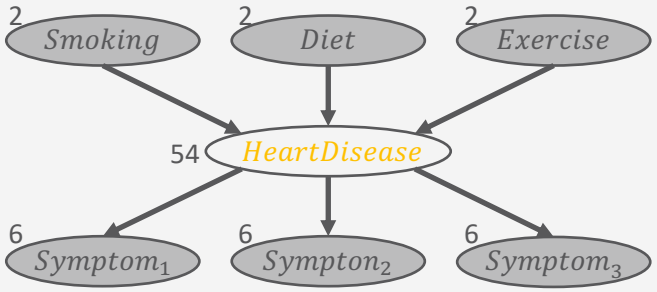
No ordering prescribed; implicitly required that all $\boldsymbol{C}_i$ visited enough
- E.g., start at a leaf, traverse the clusters by depth-first search

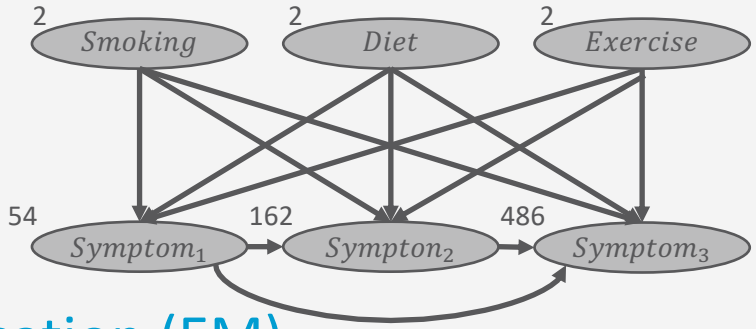
IPF-JT

Yee Whye Teh and Max Welling: On Improving the Efficiency of the Iterative Proportional Fitting Procedure. In: *AISTATS-03 Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

Parameter Learning with Hidden (Latent) Variables

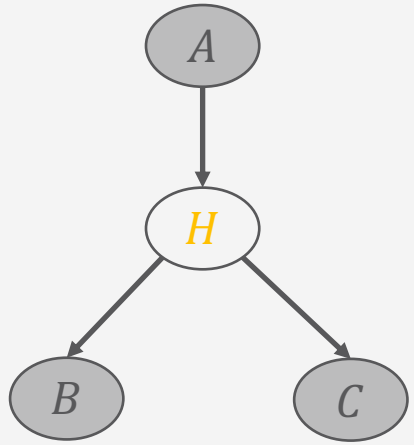
- Problem with hidden variables in model: Counting not possible as data not available
  - Avoiding hidden variables not an option: explosion of parameters to learn





- Solution: Expectation-Maximisation (EM)
  - If we had data for all the variables in the network, we could learn the parameters by using ML methods
  - If we had the parameters in the network, we could generate data for the hidden variables (by sampling + counting or estimating expected counts)

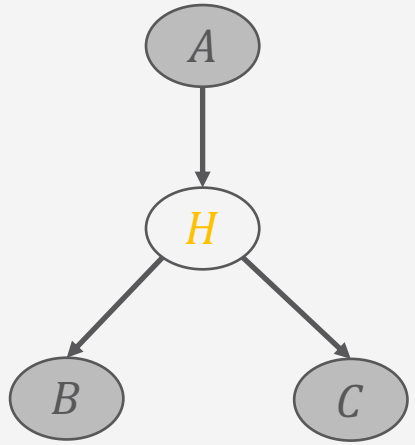
A	B	C
$true$	$false$	$true$
$false$	$true$	$true$
$true$	$true$	$false$
	$\vdots$	



EM: General Idea

- *Initialisation*: Set parameters to "invented" (e.g., randomly generated) values
- *Loop*: Refine parameters by cycling through two basic steps *until* likelihood of given data plateaus
  - Expectation (E):
    Update data with predictions generated via the current model (explicitly sample, or compute expected counts $\widehat{N}$)
    - Example: $\widehat{N}(H = i) = \sum_{j=1}^{N} P(H = i \mid a_j, b_j, c_j)$
      - Sum over all samples the probability of H being i given j'th sample a_j, b_j, c_j
  - Maximisation (M):
    Given the updated data, update the model parameters using MLE
    - Same step as when learning parameters for fully observable networks
      - Use relative frequencies in BNs, IPF procedure for factor-based models

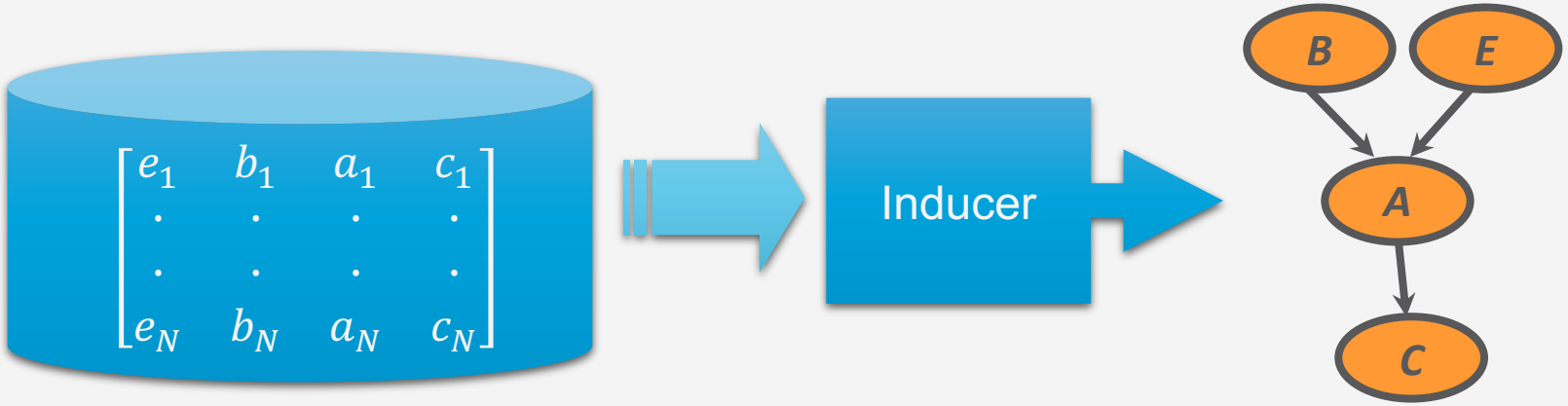
A	B	C
$true$	$false$	$true$
$false$	$true$	$true$
$true$	$true$	$false$
	$\vdots$	



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



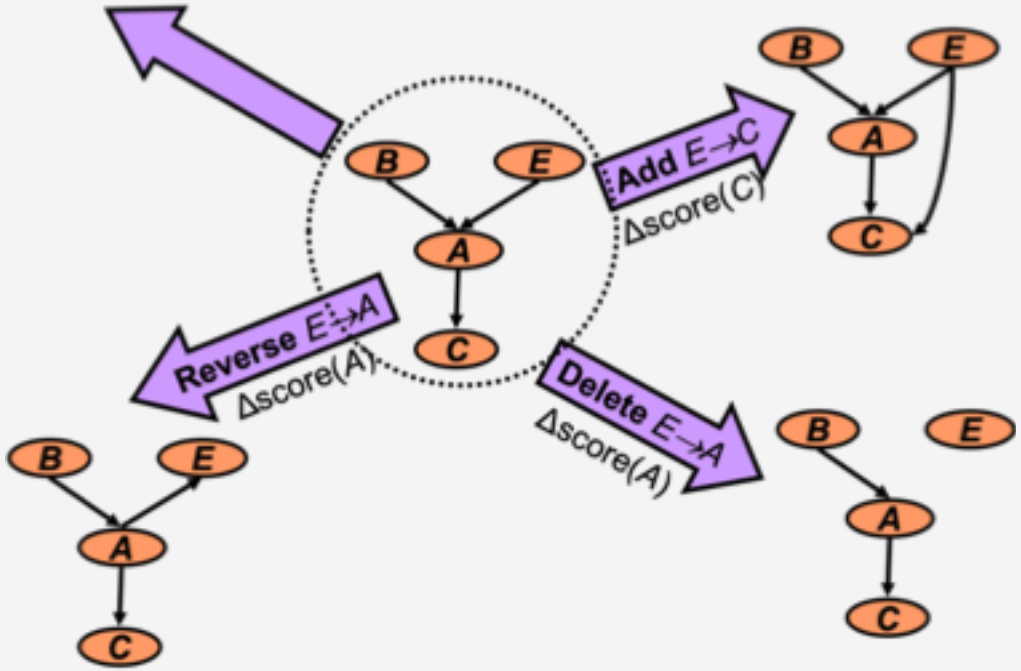
Data d

Some of the following slides from an AI course "Graphical models" by Burgard/De Raedt/Kersting/Nebel

Model selection

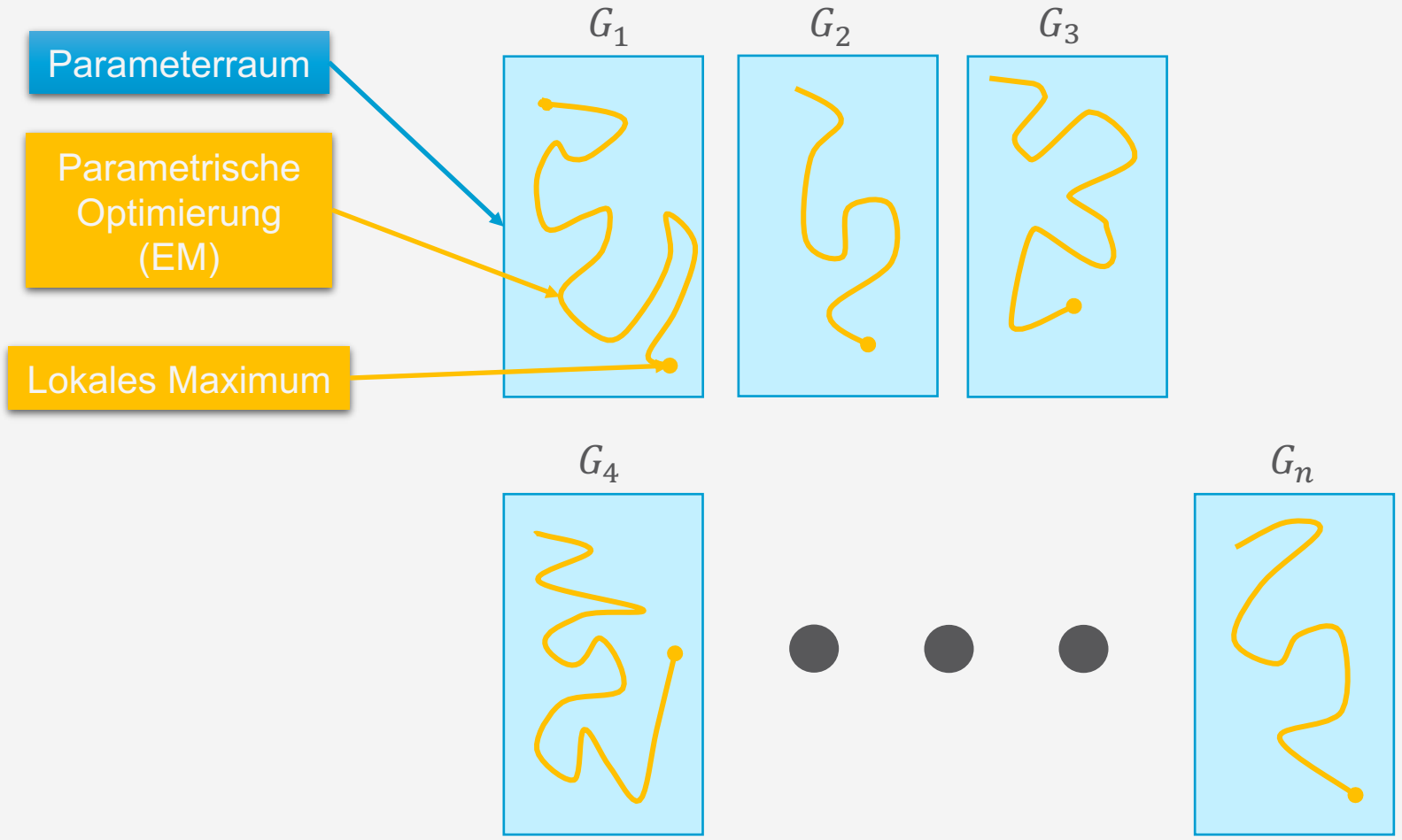
- Goal: Select the best network structure
- Input: Training data, scoring function
- Output: A network that maximises the score

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

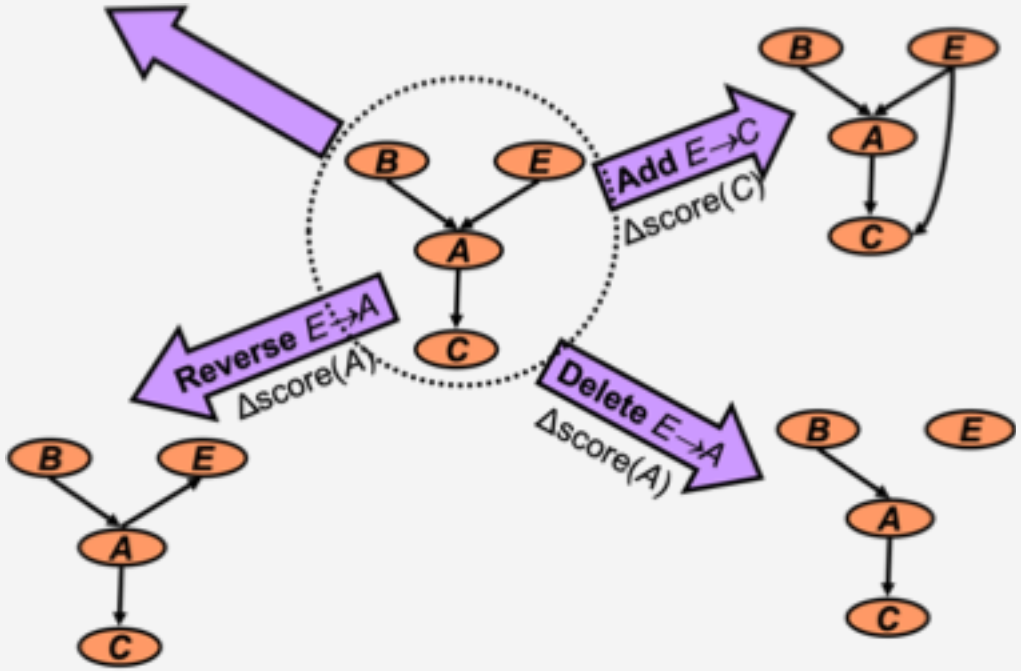
- Perform EM for each candidate graph
- Computationally expensive
  - Parameter optimisation via EM – non-trivial
  - Need to perform EM for all candidate structures
  - Spend time even on poor candidates
  - Data might be incomplete



Structure Learning: Incomplete Data

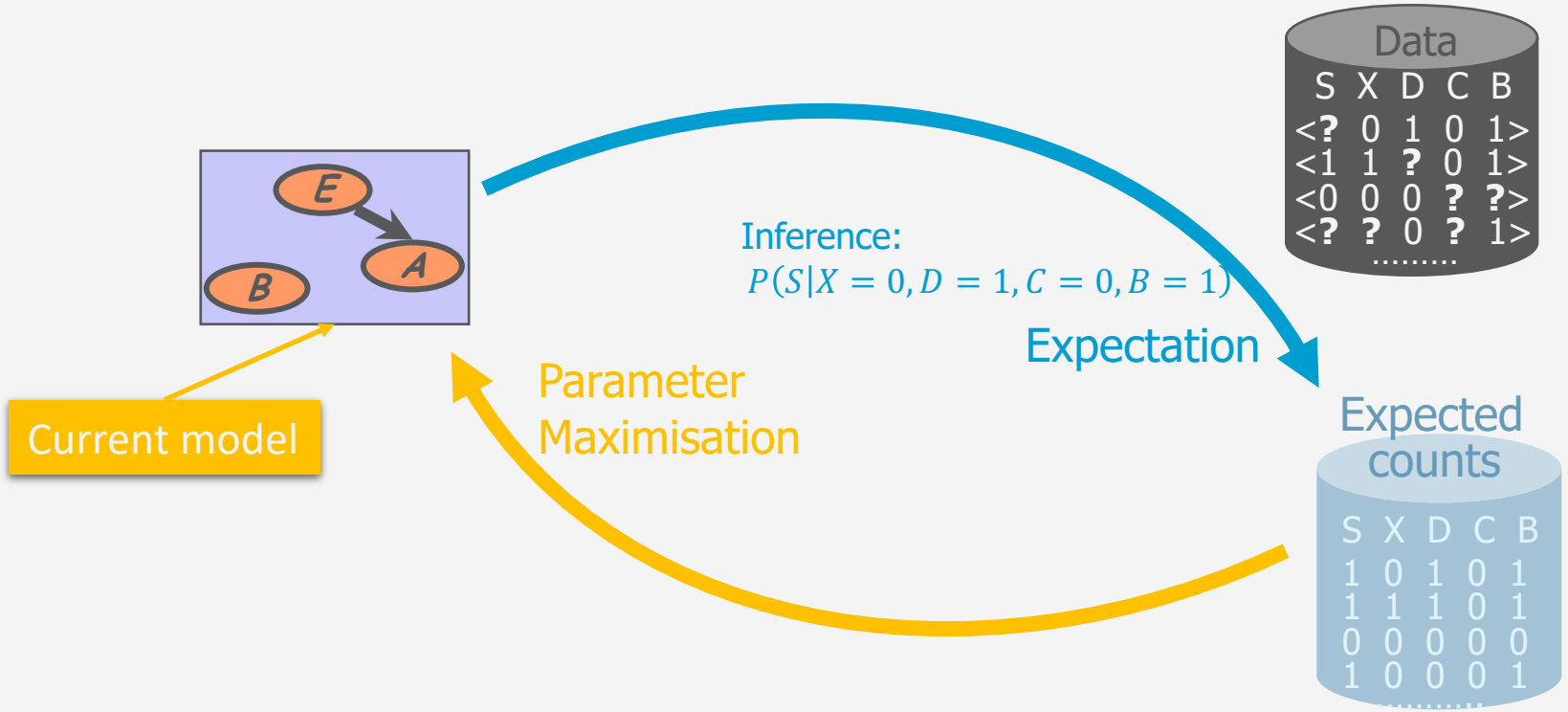
- There might be hidden variables
  - No data available for the hidden variables
  - Search space becomes that much larger

- 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



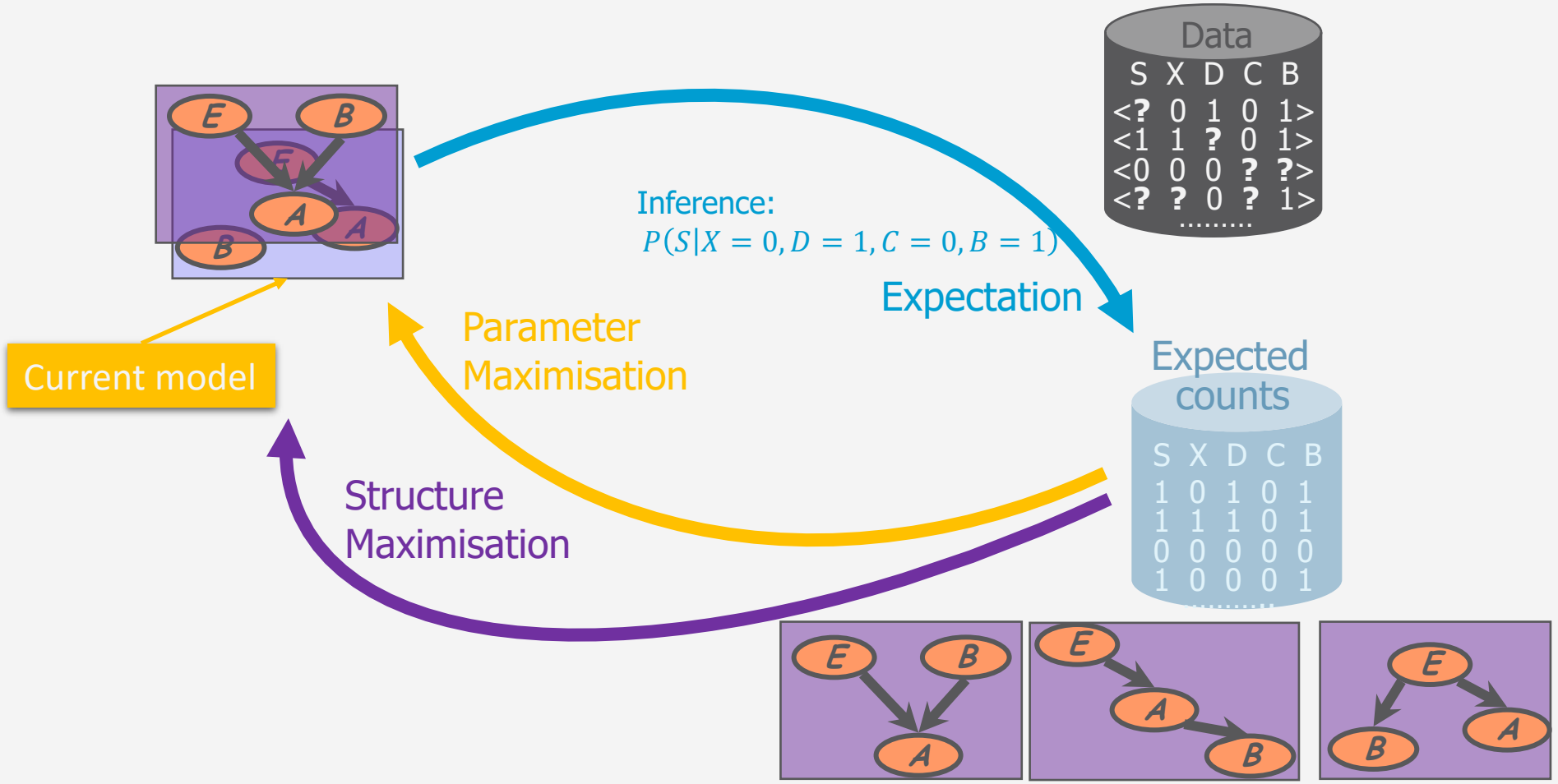
EM

- EM-Algorithm:
  Iterate
  until convergence



Structural EM

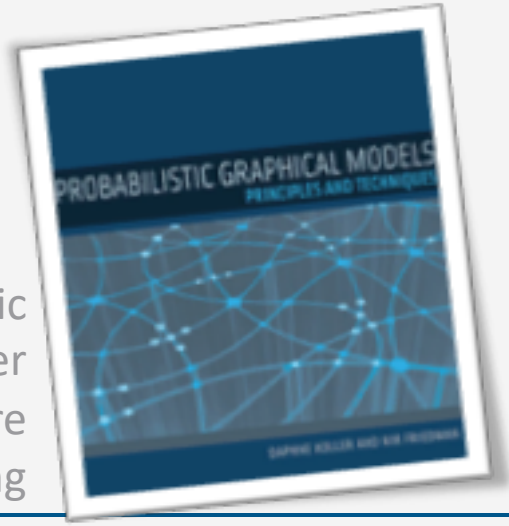
- SEM-Algorithm:
  Iterate
  until convergence



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 $P_{\boldsymbol{R}}$ MLE-style
  - Find which random variables are independent of each other, delete edges accordingly
  - Alternative: hypothesis tests to not learn $P_{\boldsymbol{R}}$ 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. Lifted Learning

A. *Overview of (propositional) learning*
- Parameter and structure learning

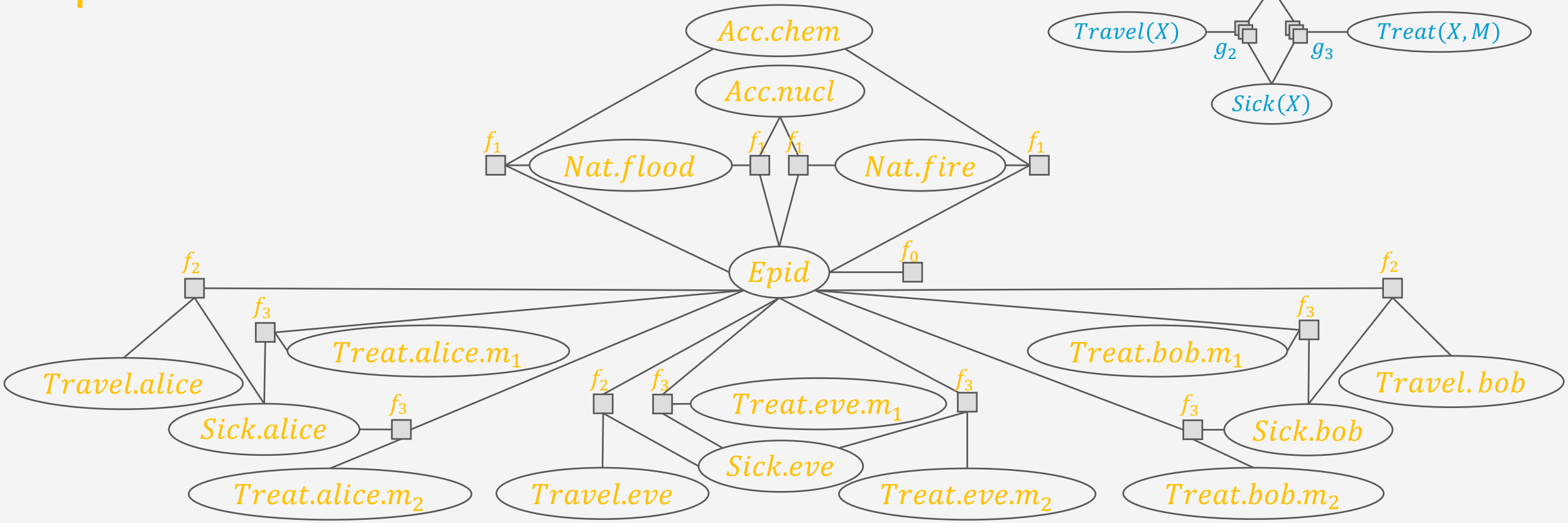
B. *Lifted encoding of propositional models*
- Colour passing

C. *Relation(al) learning*
- First-order inductive learning, decision tree representation
- Relational dependency network learning
- Changing domains

D. *Approximating Symmetries*
- Evidence and model-based approaches, local structure

Compression

Input



Goal

Colour Passing

- Based on the idea of (loopy) belief propagation
  - Exploit computational symmetries
  - Compress graph whenever nodes would send identical messages
  - Use compressed graph for probabilistic inference

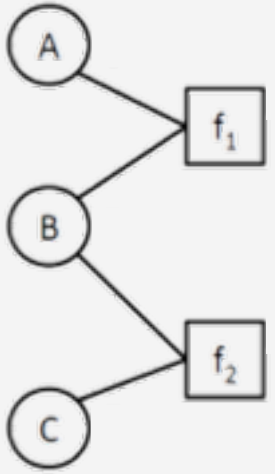
→ Colour passing algorithm for compression

- Make symbolic message passes, sending colours instead of computed messages
  - Less computational effort

Parag Singla and Pedro Domingos: Lifted First-order Belief Propagation. In *AAAI-08 Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 2008.
Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting Belief Propagation. In *UAI-09 Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 2009.
Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. In *Machine Learning*. 92(1):91-132, 2013.

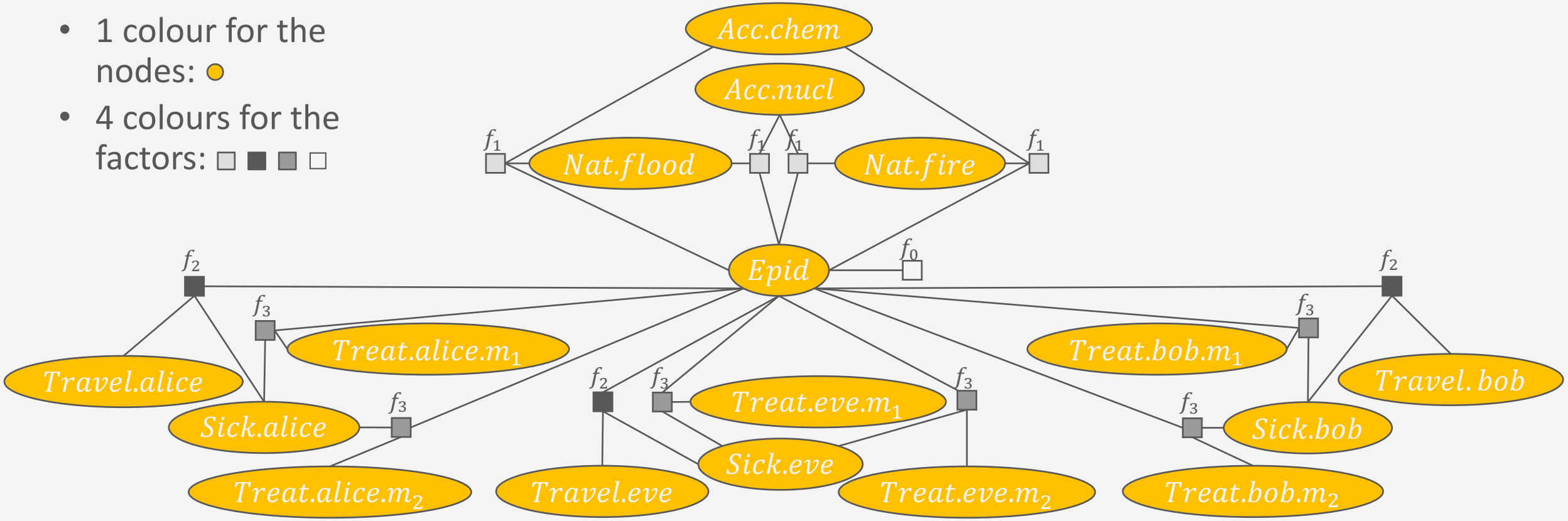
Compression: Pass the Colours Around

- 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**



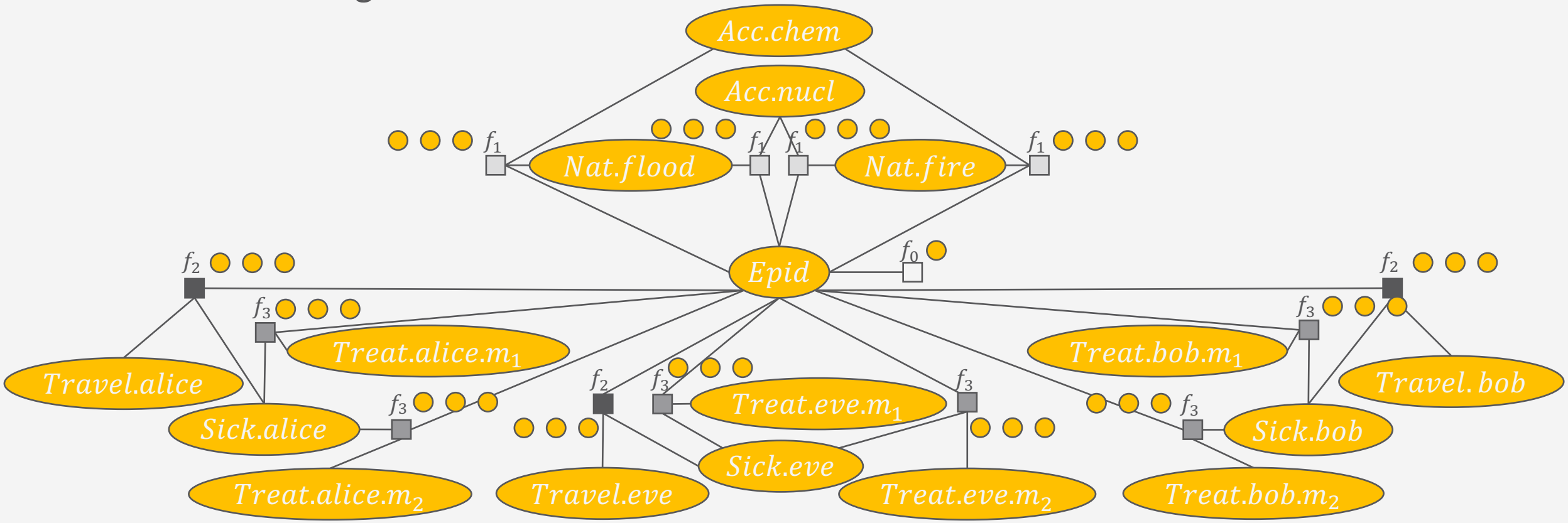
Compression

- Initialisation: Colour nodes and factors
  - 1 colour for the nodes: ●
  - 4 colours for the factors: □ ■ ■ □



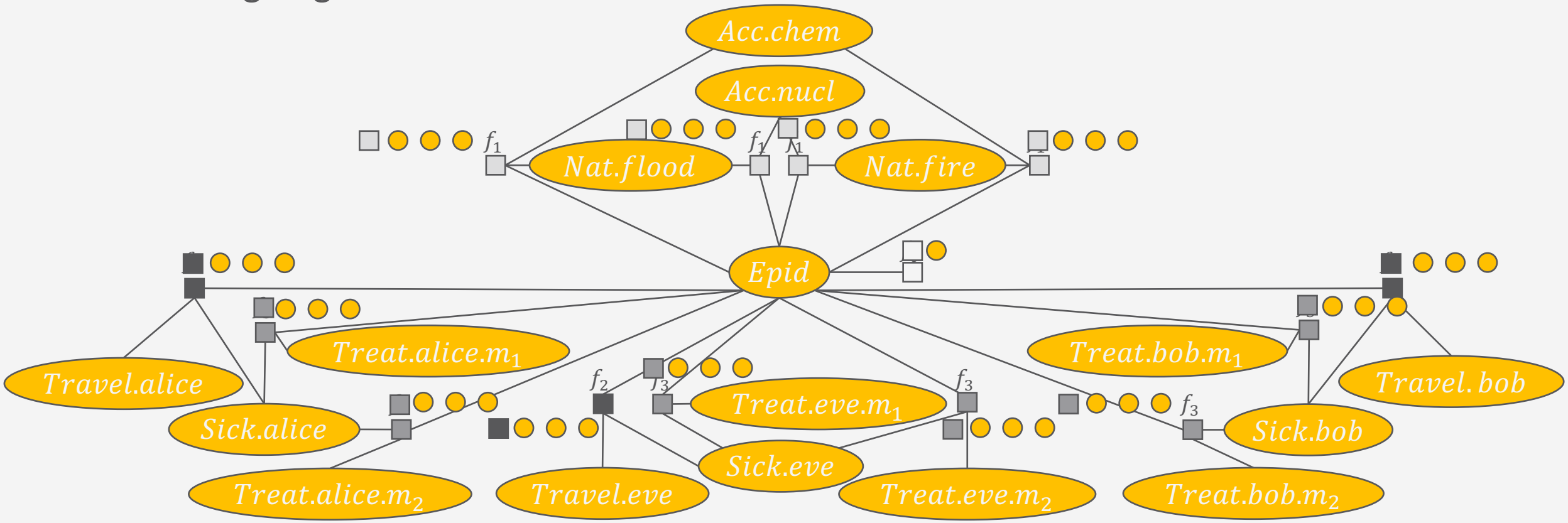
Compression

1. Factors collecting colours from nodes



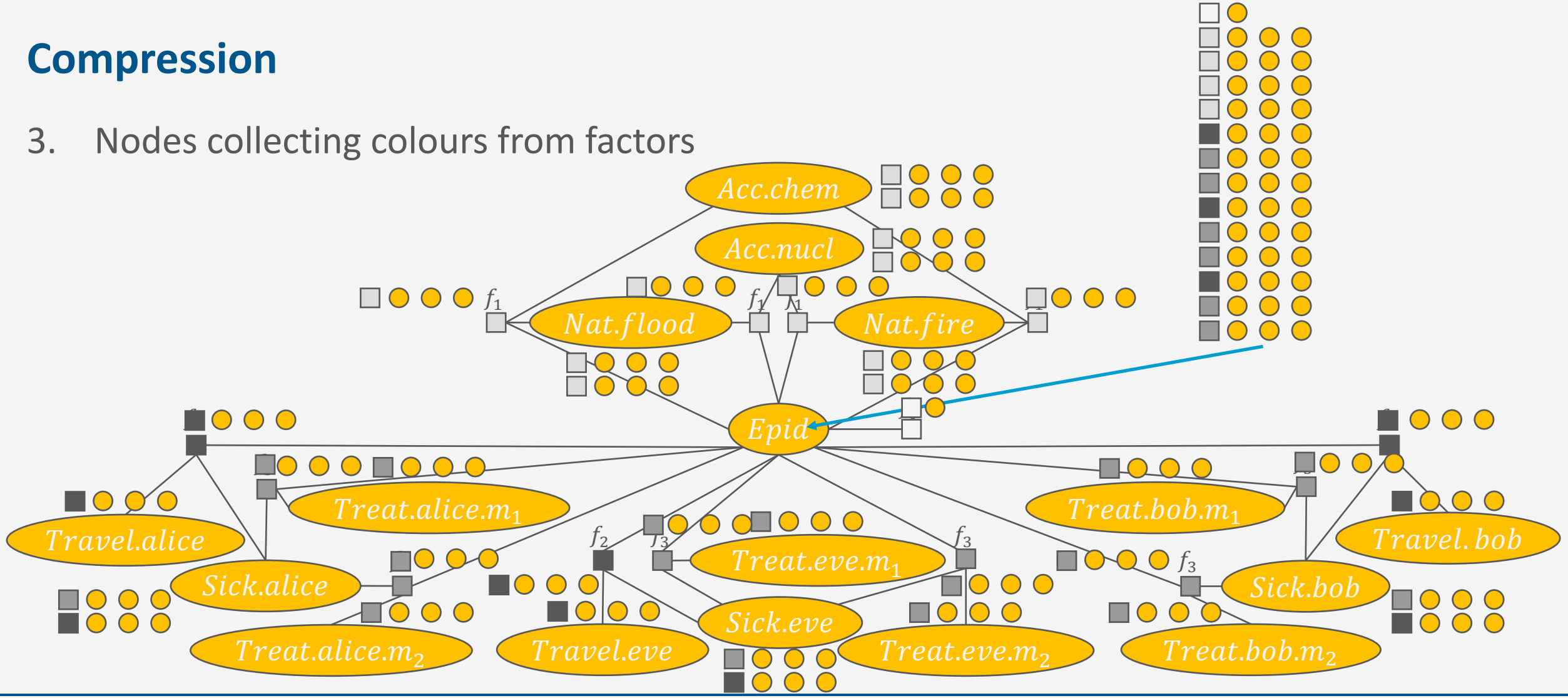
Compression

2. Factors signing their own colours to the collected ones



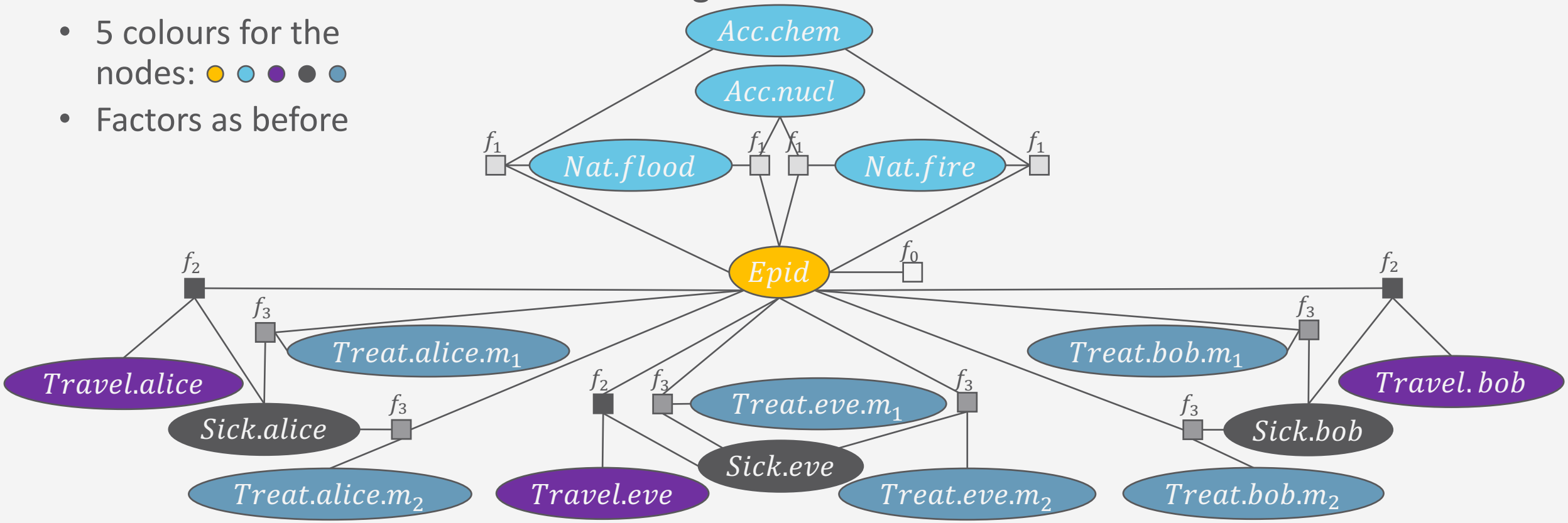
Compression

3. Nodes collecting colours from factors



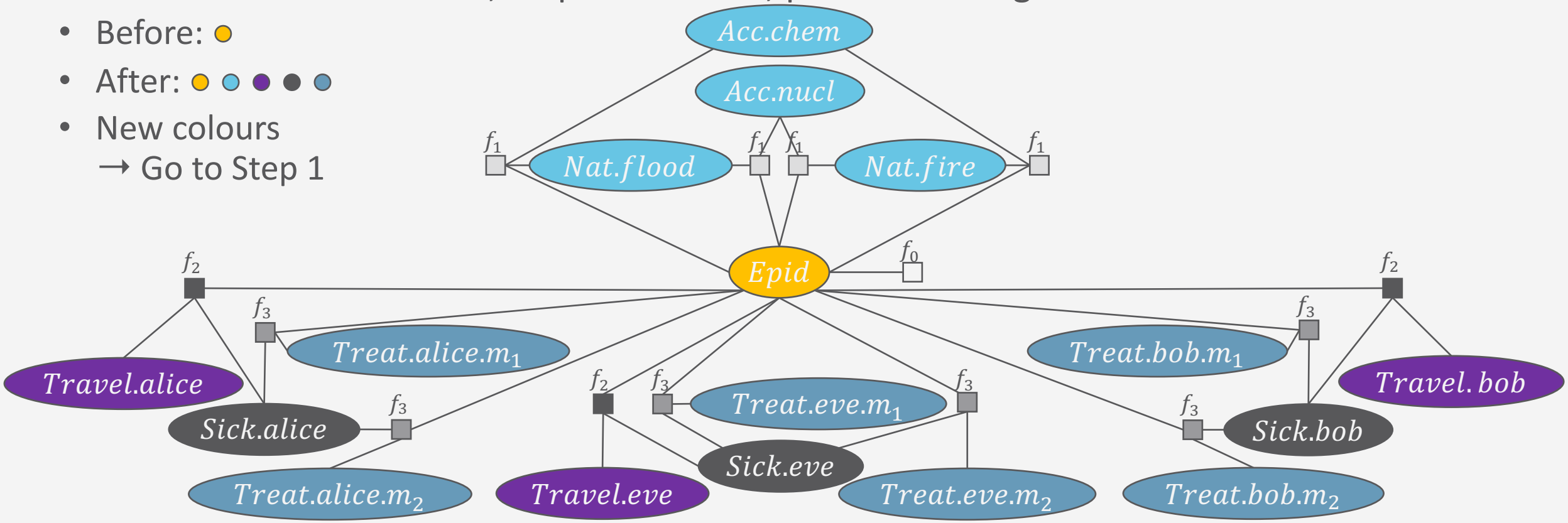
Compression

4. Recolour nodes based on collected signatures
   - 5 colours for the nodes: ● ● ● ● ●
   - Factors as before



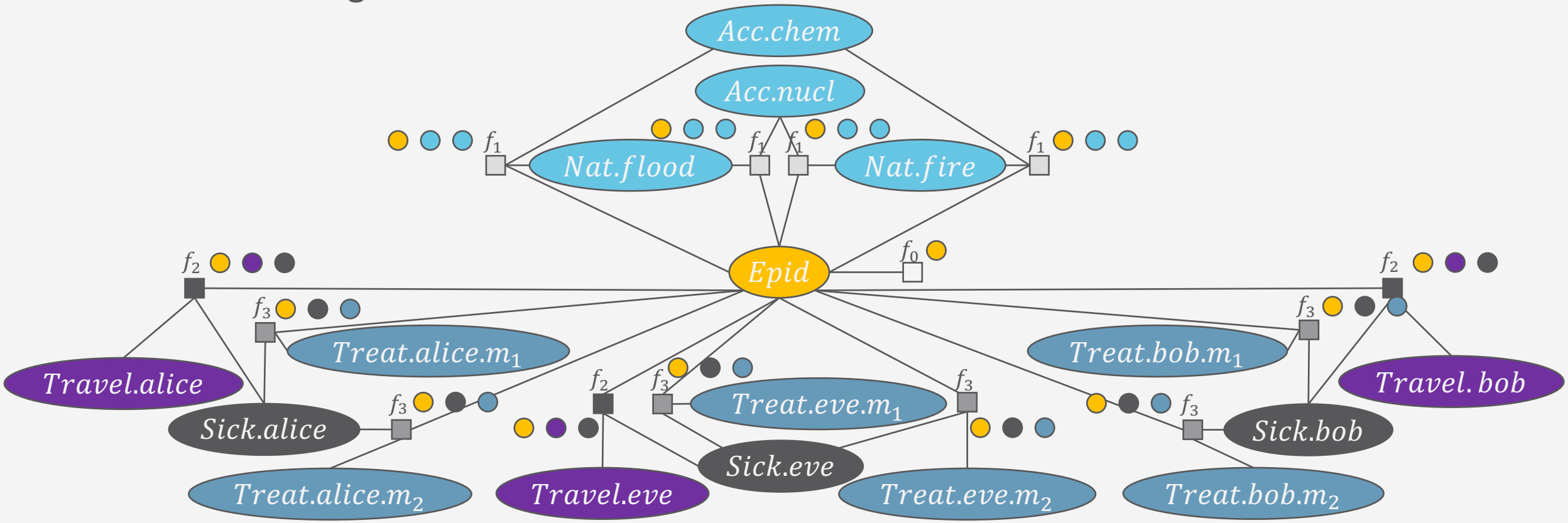
Compression

5. If no new colour created, stop. Otherwise, pass colours again.
   - Before: ●
   - After: ● ● ● ● ●
   - New colours
     → Go to Step 1



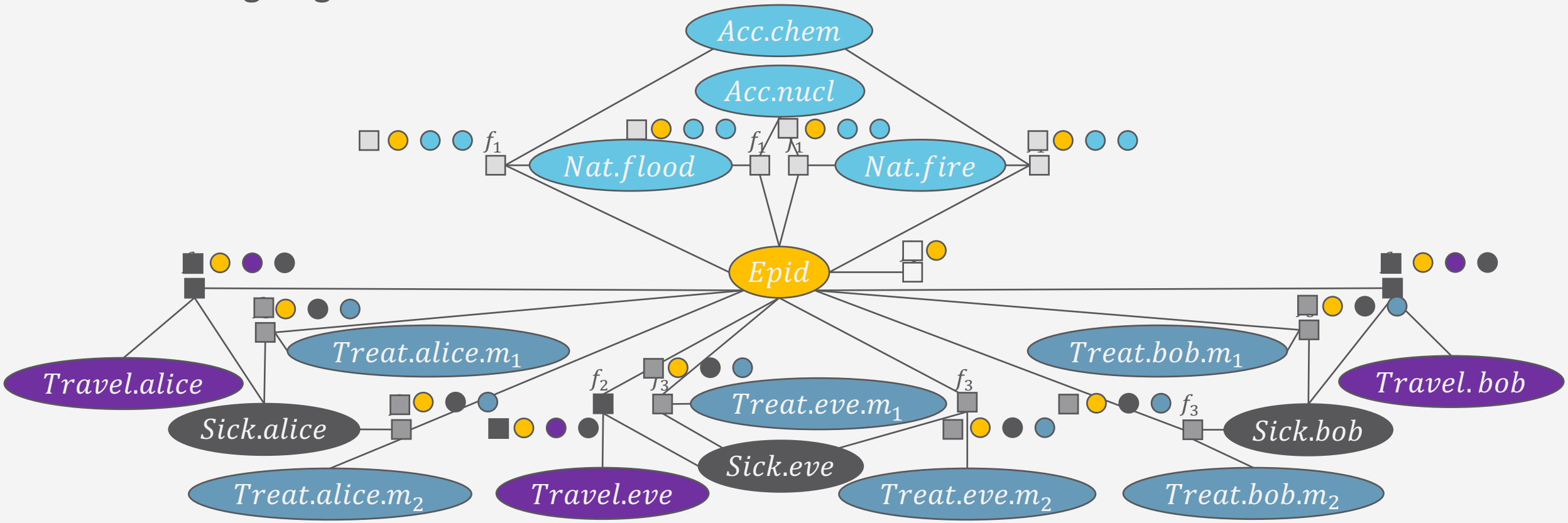
Compression

1. Factors collecting colours from nodes



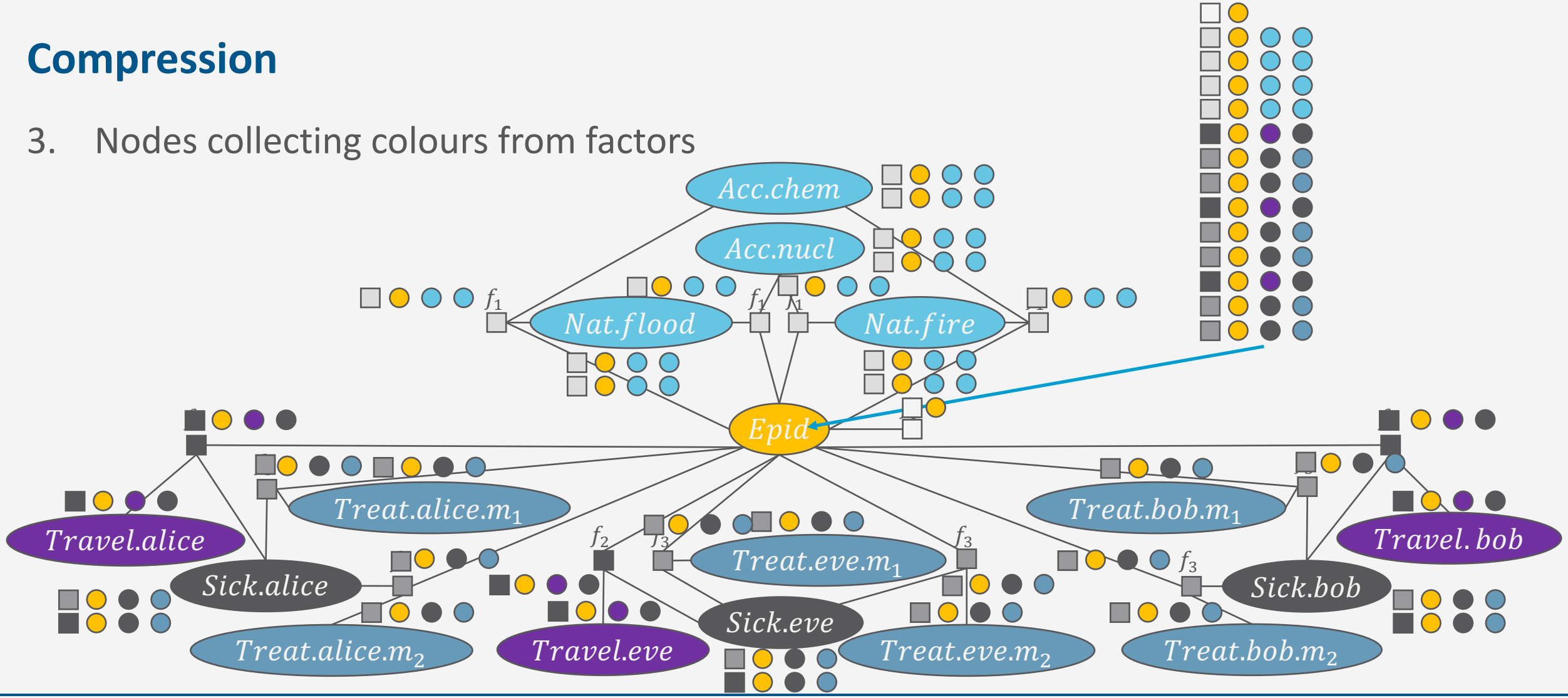
Compression

2. Factors signing their own colours to the collected ones



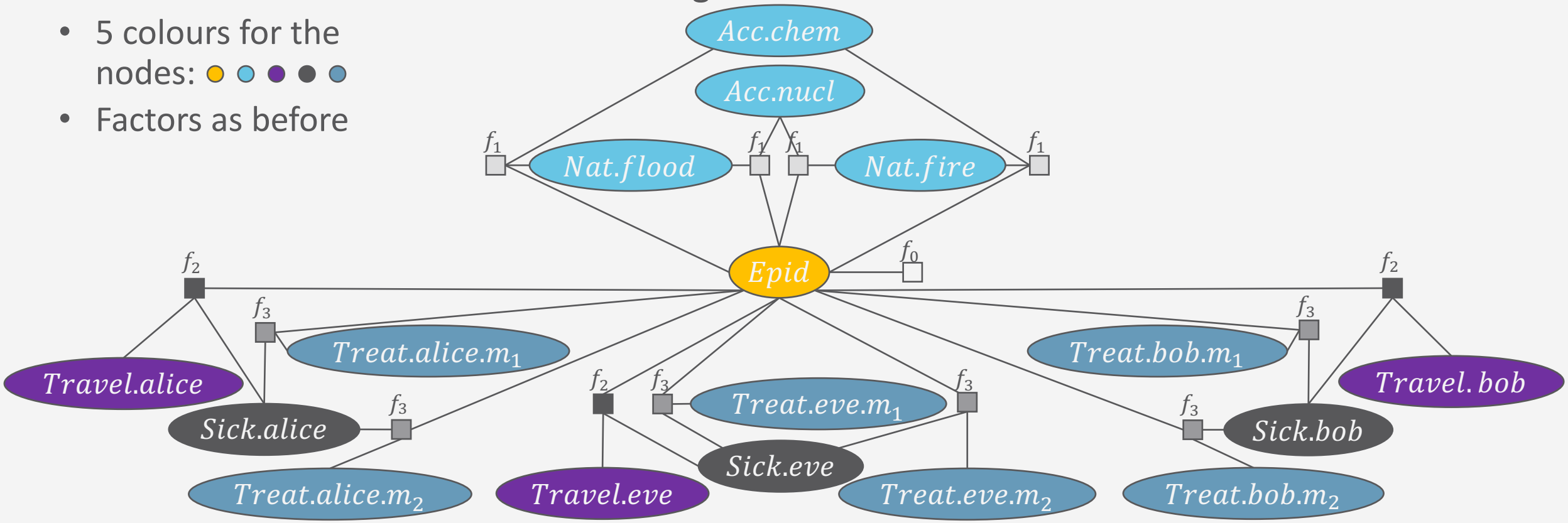
Compression

3. Nodes collecting colours from factors



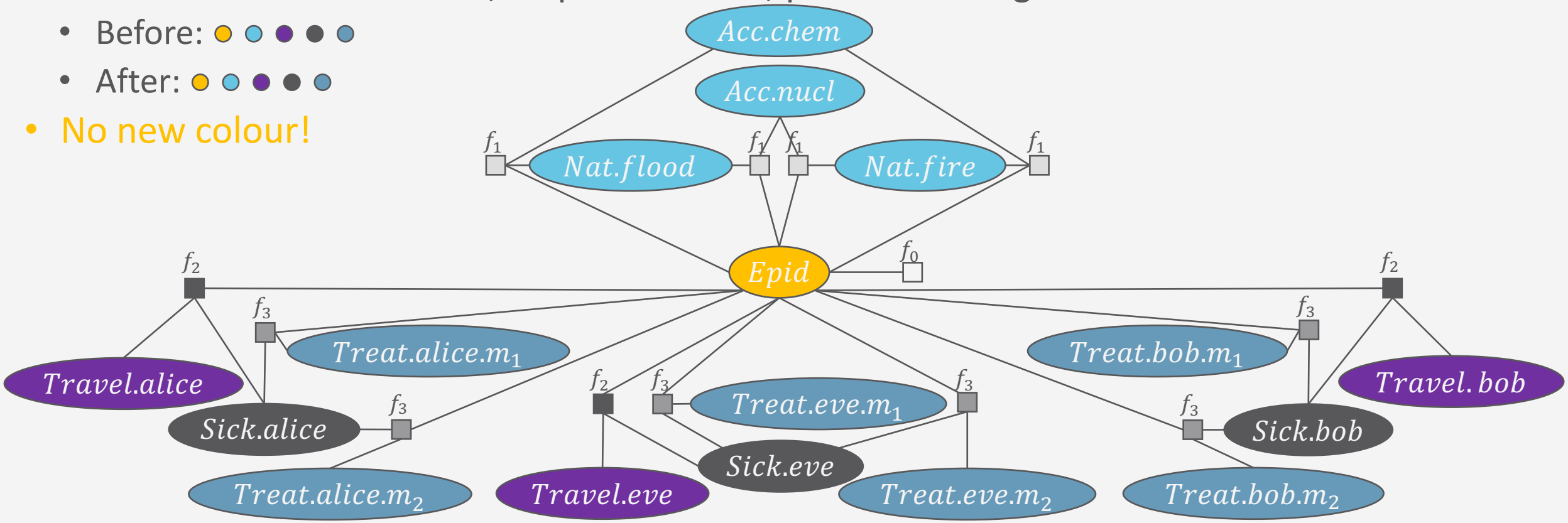
Compression

4. Recolour nodes based on collected signatures
   - 5 colours for the nodes: ● ● ● ● ●
   - Factors as before



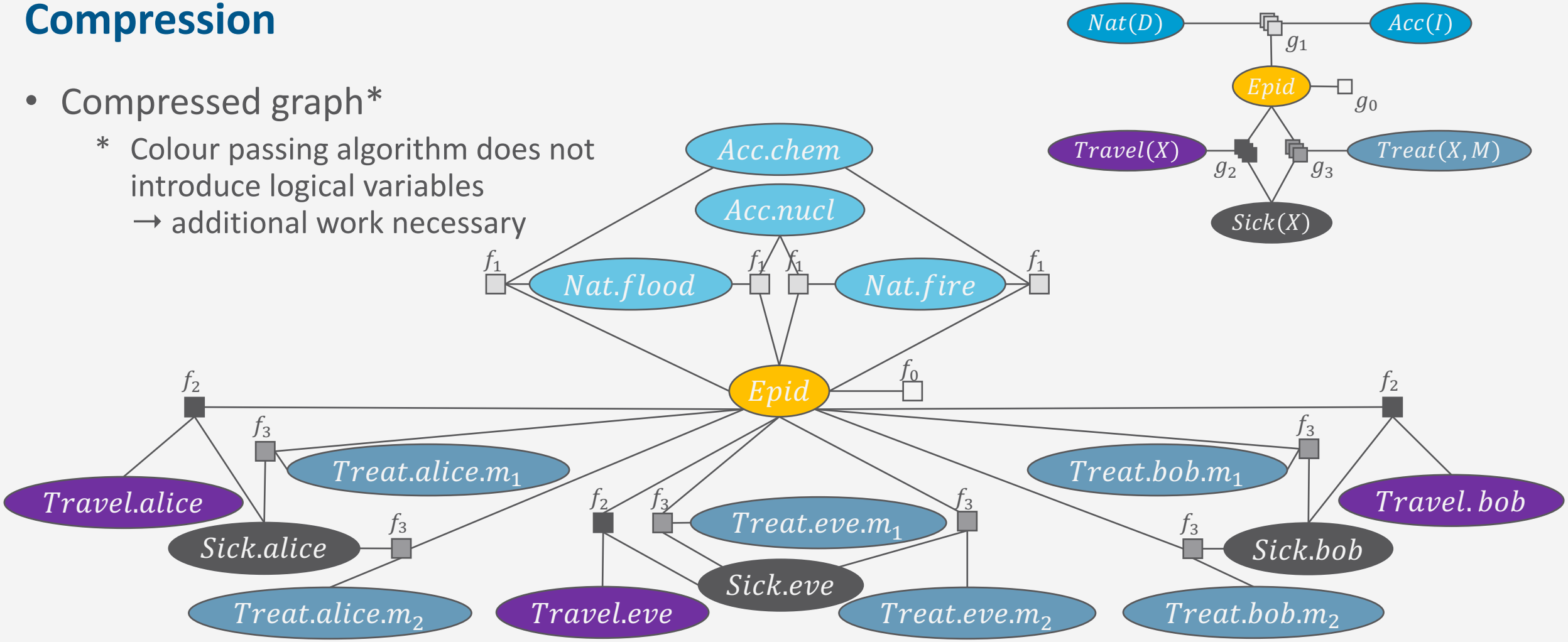
Compression

5. If no new colour created, stop. Otherwise, pass colours again.
   - Before: ● ● ● ● ●
   - After: ● ● ● ● ●
- No new colour!



Compression

- Compressed graph*
  - * Colour passing algorithm does not introduce logical variables
    → additional work necessary



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
  - Could specify an approximate version to further compress a model
    - E.g., consider $(1.0,2.0)$ and $(1.1,2.0)$ to be equivalent

Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. In *Machine Learning*. 92(1):91-132, 2013.

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

Outline: 4. Lifted Learning

A. Overview of (propositional) learning
- Parameter and structure learning

B. Lifted encoding of propositional models
- Colour passing

C. Relation(al) learning
- First-order inductive learning, decision tree representation
- Relational dependency network learning
- Changing domains

D. Approximating Symmetries
- Evidence and model-based approaches, local structure

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
  - Relation learning in logics, e.g.,
    - First-order inductive learning (FOIL)
    - First-order logical decision trees (FOLDTs)
      - Combined with weights/probabilities
  - 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)

Knowledge-based Inductive Learning

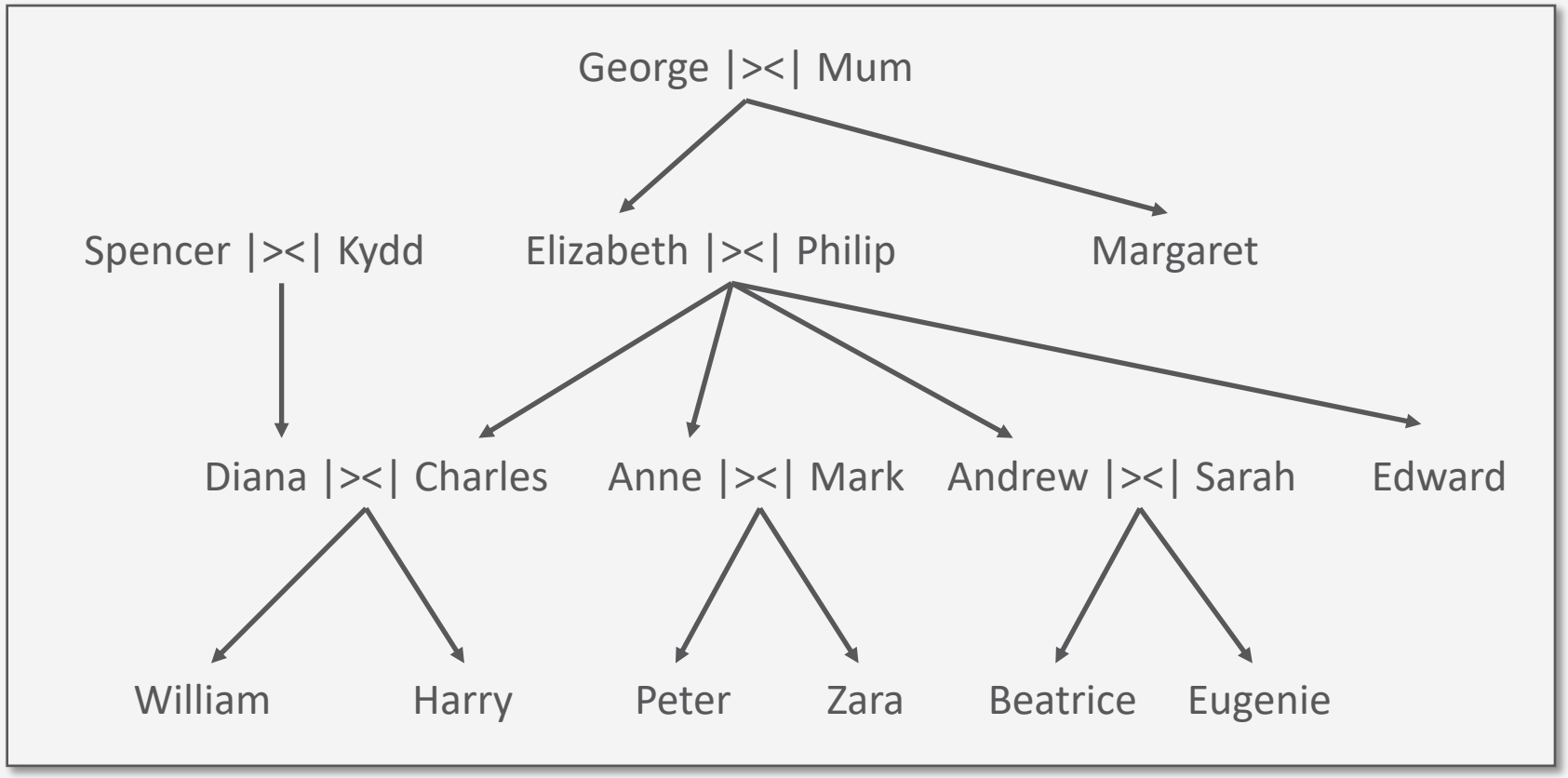
- Logic perspective on learning
  - Examples are composed of descriptions and classifications
  - Objective is to find a hypothesis that explains the classification of the examples, given their descriptions
    - $\text{Hypothesis} \land \text{Descriptions} \models \text{Classifications}$
- Knowledge-based inductive learning
  - *Background knowledge* helps to explain examples
    - $\text{Background} \land \text{Hypothesis} \land \text{Descriptions} \models \text{Classifications}$
    - 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., $Grandparent$, $BrotherInLaw$, $Ancestor$

John Ross Quinlan: Learning Logical Definitions from Relations. In *Machine Learning*, 1990.

A Not Up-to-date Example



Example: $Grandparent$

Background ∧ Hypothesis ∧ Descriptions ⊨ 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:

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

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

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

    - Grandparent is now reduced to

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

- 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 $\text{New}-\text{Clause}(examples, target)$ **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**
    $l \leftarrow \text{Choose}-\text{Literal}(\text{New}-\text{Literals}(clause), extended)$
    append l to the body of clause
    $extended \leftarrow$ set of examples created by applying
      $\text{Extend}-\text{Example}$ to each example in $extended$ for l
  return $clause$

FOIL: new clause

FOIL: New Literals

- $\mathrm{New{-}Literals}$: 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

- $\mathrm{Choose{-}Literal}$: 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

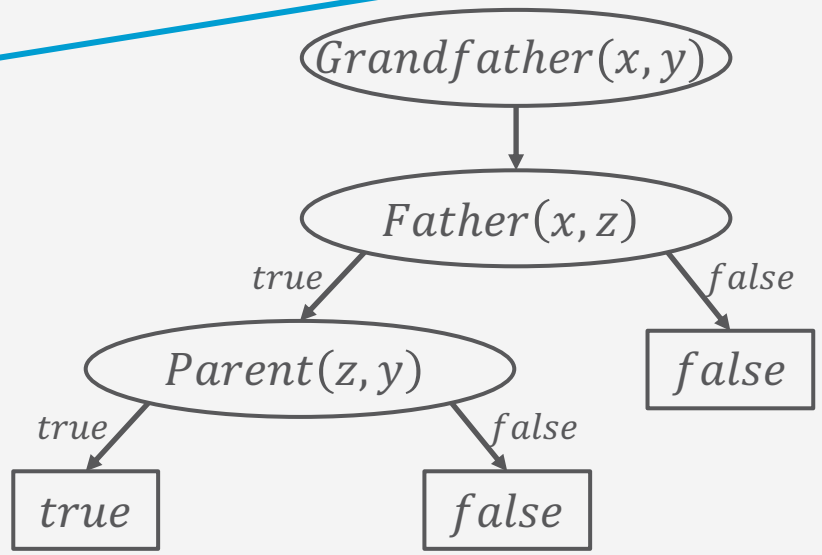
- The way $\mathrm{New}-\mathrm{Literal}$ changes the clauses leads to a very large branching factor
  - Improve performance by using type information
    - E.g., $Parent(x, n)$ where x is a person and n is a number
  - Ockham's razor to eliminate hypotheses
    - If a clause becomes longer than the 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



Hendrik Blockeel and Luc De Raedt: Top-down Induction of First-order Logical Decision Trees. In *Artificial Intelligence*, 1998.

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

Hendrik Blockeel and Luc De Raedt: Top-down Induction of First-order Logical Decision Trees. In *Artificial Intelligence*, 1998.

FOLDT: Example

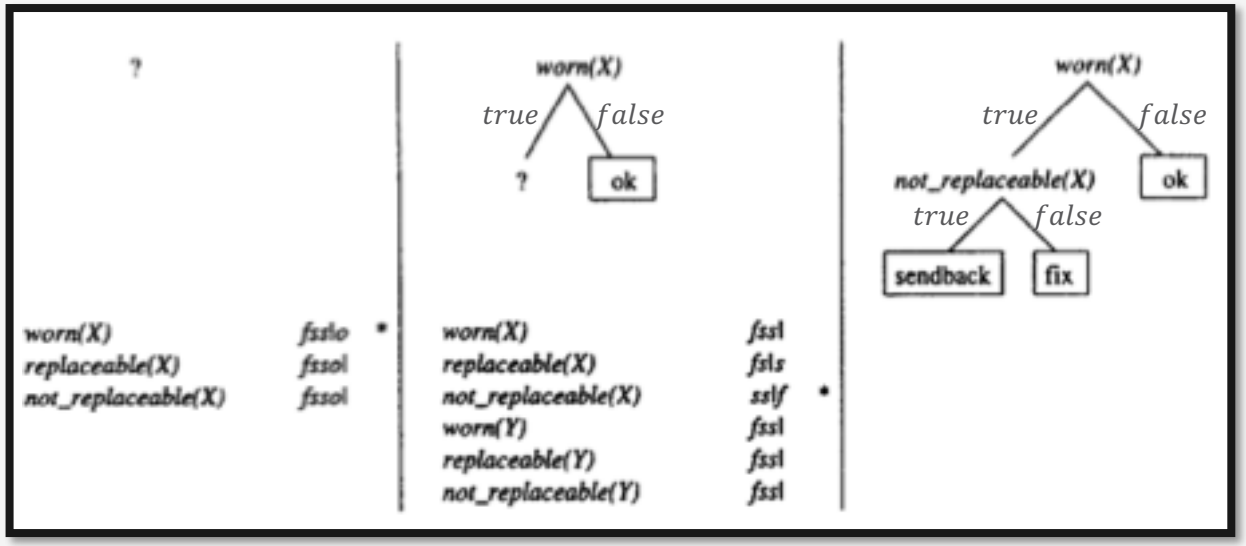
- Idea of what is to learn:
  - Check a machine with parts X
  - If machine contains worn parts that cannot be replaced by engineer, send back to manufacturer
  - If all worn parts can replaced, then fix it
  - No worn parts, ok
- Learning progress: to the right below
- Resulting logic program:

```
class(ok)       ← ∀X : ¬worn(X)
class(sendback) ← ∃X : worn(X) ∧ not_replaceable(X)
class(fix)      ← ∃X : worn(X) ∧
                  ∀Y : (¬worn(Y) ∨ ¬not_replaceable(X))
```

Input examples, background knowledge

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

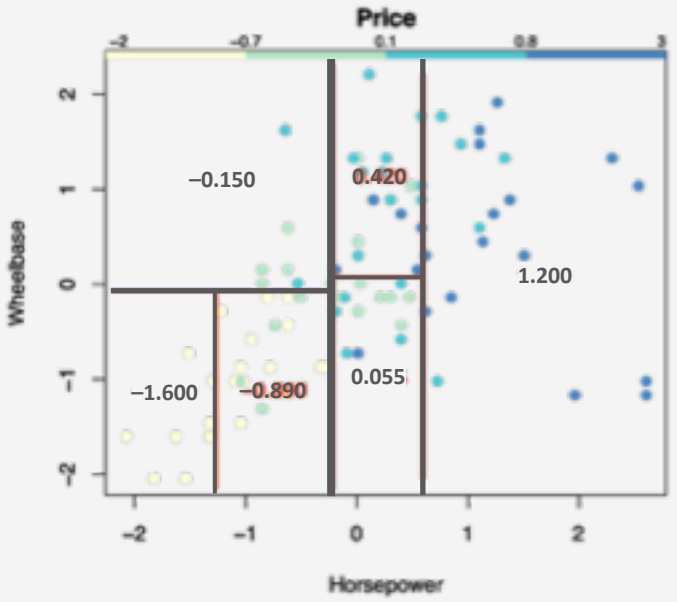
Background knowledge
replaceable(gear)
replaceable(chain)
not_replaceable(engine)
not_replaceable(wheel)

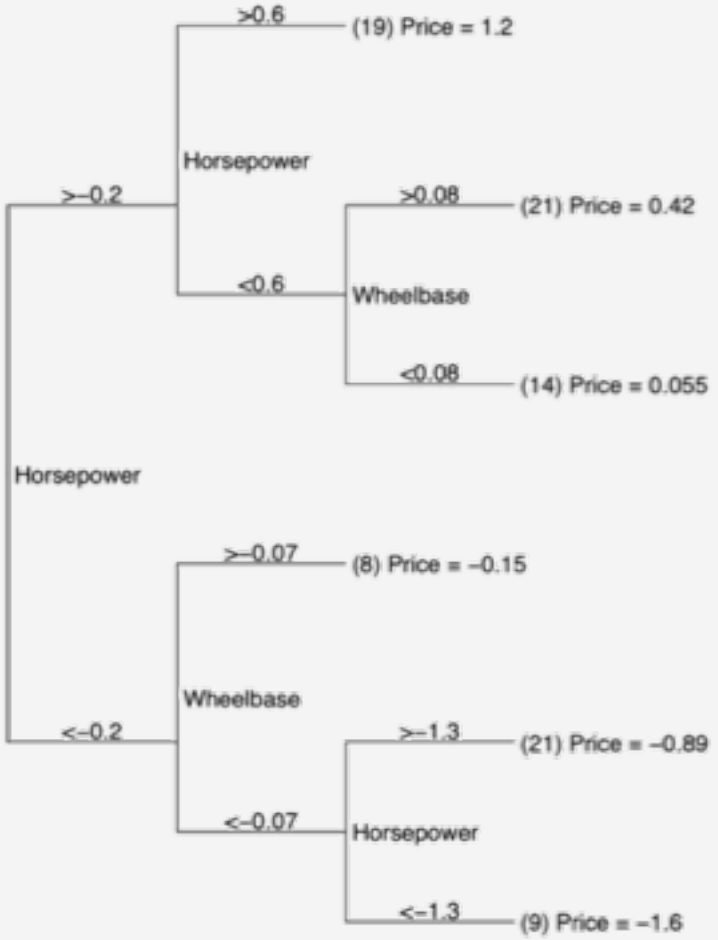


Hendrik Blockeel and Luc De Raedt: Top-down Induction of First-order Logical Decision Trees. In *Artificial Intelligence*, 1998.

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



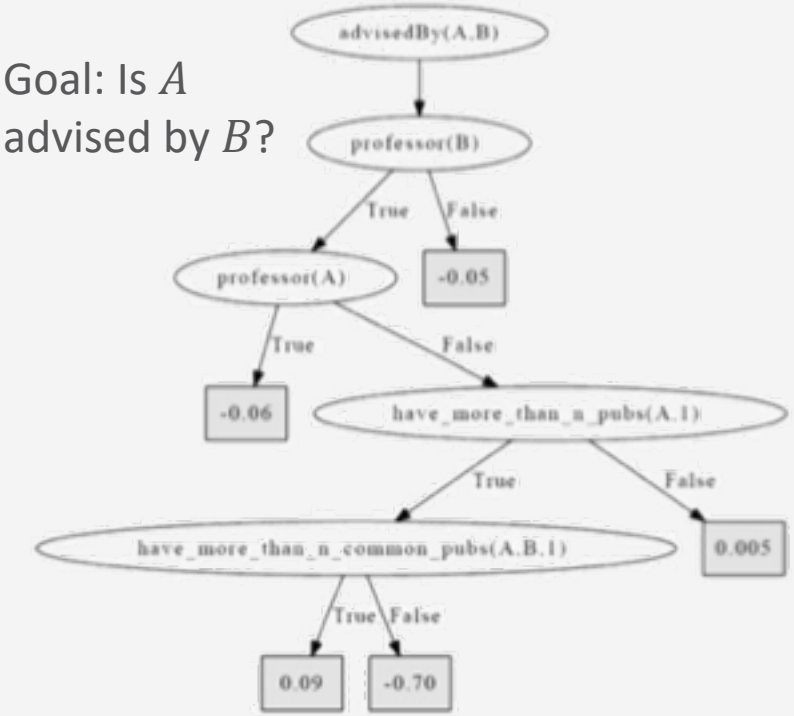


http://www.stat.cmu.edu/~cshalizi/350-2006/lecture-10.pdf

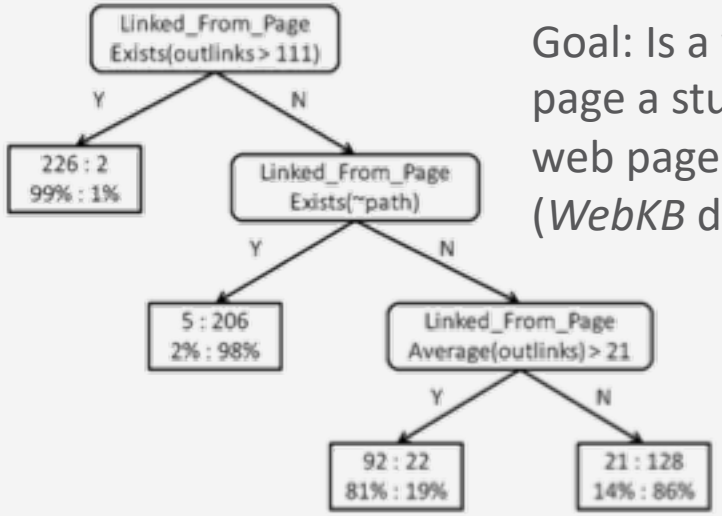
Relational Regression & Probability Trees

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

Goal: Is A advised by B?



- Relational probability tree (RPT)
  - ≈ FOLDT with *probability distributions* in leaves



Goal: Is a web page a student web page? (*WebKB* data set)

There are some differences what they allow inner nodes to be
- Not important to grasp the general idea

Left 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.
Right figure: Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay: Learning Relational Probability Trees. In *SIGKDD-03 Proceedings*, 2003. // *WebKB* data set: http://www.cs.cmu.edu/~webkb/

Relational Dependency Networks

Learning approximate models

Learn Approximate Models

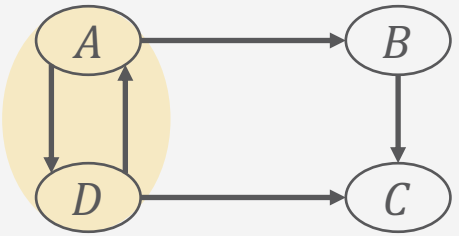
- Relational dependency networks (RDNs)
  - Using RPTs for local distributions in RDNs
  - Boosted learning: Learn a set of distributions to approximate a local distribution
    - Set of RRTs for local distributions in RDNs
  - Based on approximate propositional model of dependency networks (DN)
- Next 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) $P\left(R_k | parents(R_k)\right)$ 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

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



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

David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie: Dependency Networks for Inference, Collaborative Filtering, and Data Visualisation. In *Journal of Machine Learning Research*, 2000.

Relational Dependency Networks

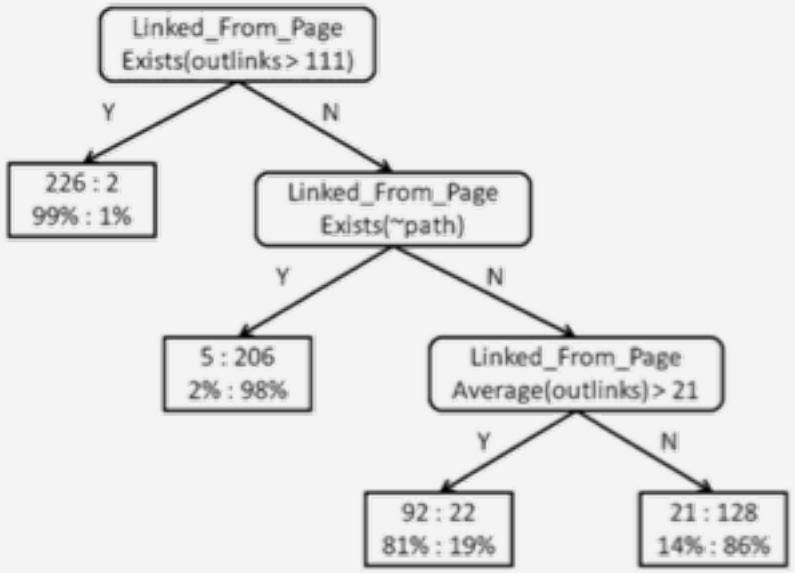
- Relational aspects explicitly modelled in DN
  - Relational databases as original motivation and backend for algorithms; logic perspective here
- Represent joint distribution as a product of (conditional) marginals over ground atoms
  - Inference by grounding and unrolling the model such that we have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

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

Jennifer Neville and David Jensen: Relational Dependency Networks. In *Journal of Machine Learning Research*, 2007.
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.

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

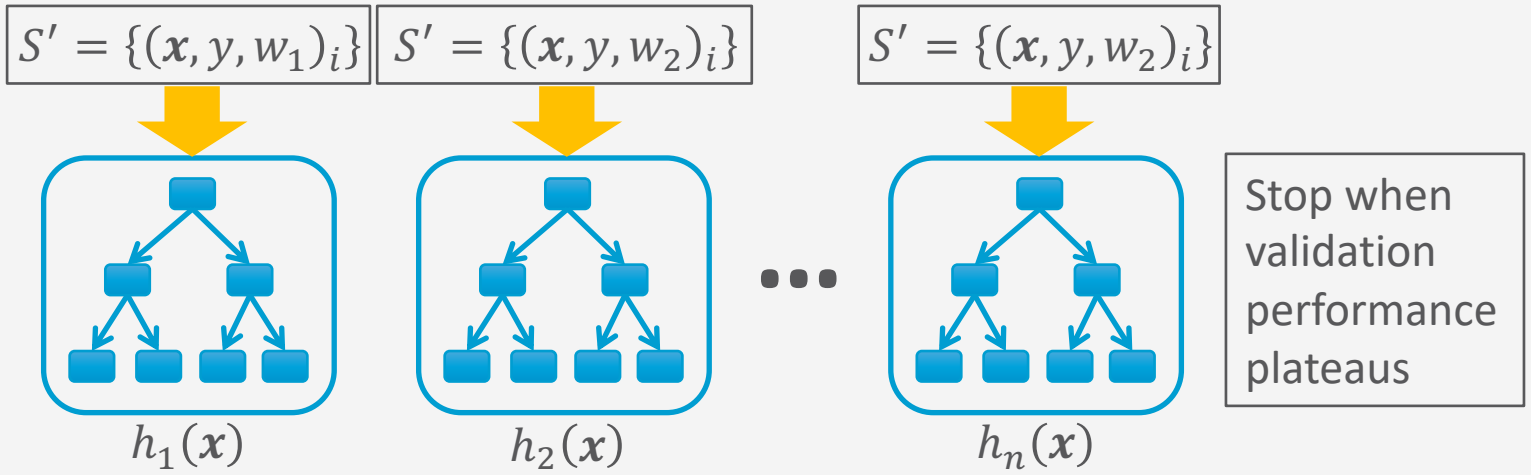
Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay: Learning Relational Probability Trees. In *SIGKDD-03 Proceedings*, 2003.

Boosting Idea

- *Ensemble* of (simple) classifiers ($\boldsymbol{x}$: 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 $h(\boldsymbol{x})$
    - Weighted sum: $h(\boldsymbol{x}) = \sum_i \alpha_i h_i(\boldsymbol{x})$
  - E.g., *AdaBoost* with decision trees

Training loop:
- Train classifier $h_i(\boldsymbol{x})$ 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'

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



https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

Functional-gradient Ascent

- Models given by $\frac{\exp \psi(y;\boldsymbol{x})}{\sum_{y'} \exp \psi(y';\boldsymbol{x})}$
- 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
$$\psi_m = \psi_0 + \Delta_1 + \cdots + \Delta_m$$
  - Δ_m is the functional gradient at iteration m and given by
$$\Delta_m = \eta_m \cdot E_{x,y}\left[\frac{\partial}{\partial \psi_{m-1}} \log P(y|\boldsymbol{x};\psi_{m-1})\right]$$
    - 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

Thomas G. Dietterich, Guohua Hao, and Adam Ashenfelter: Gradient Tree Boosting for Training Conditional Random Fields. In Journal of Machine Learning Research, 2008.

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 $y_i; \boldsymbol{x}_i$ (conditioned on potential from previous iteration)

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

- Set of $\Delta_m(y_i; \boldsymbol{x}_i)$ 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 $\sum_i \big(h_m(y_i; \boldsymbol{x}_i) - \Delta_m(y_i; \boldsymbol{x}_i)\big)^2$

  - 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

Thomas G. Dietterich, Guohua Hao, and Adam Ashenfelter: Gradient Tree Boosting for Training Conditional Random Fields. In Journal of Machine Learning Research, 2008.

Functional Gradient for RDNs

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

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

  - $\psi(y; \boldsymbol{x})$ 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

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

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

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

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

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.

Functional Gradient for RDNs

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

- Functional gradient for y_i of example i

$$\Delta_m(y_i; \boldsymbol{x}_i) = \frac{\partial \log P(y_i; \boldsymbol{x}_i)}{\partial \psi(y_i = 1; \boldsymbol{x}_i)}$$

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

Indicator function that returns 1 if $a_i = 1$ and 0 otherwise

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

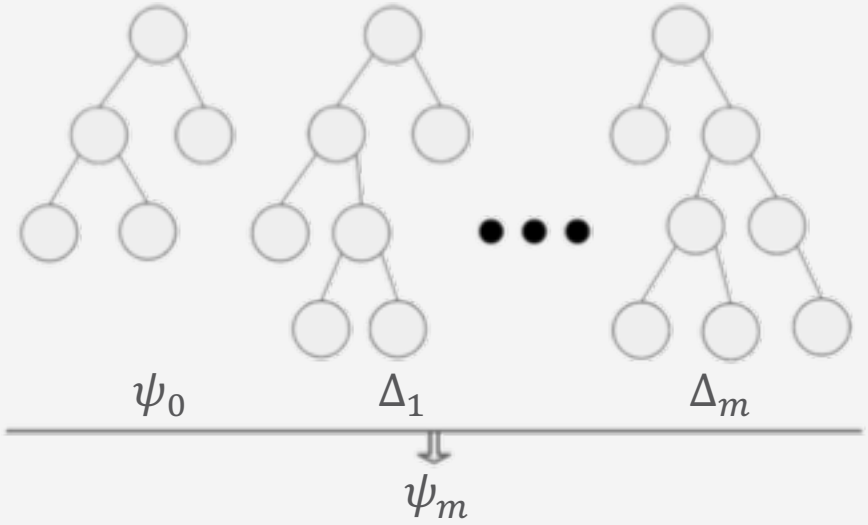
  - Gradient at each example: adjustment required for the probabilities to match the observed value for that example
  - Use $y_i; \boldsymbol{x}_i$ and $\Delta_m(y_i; \boldsymbol{x}_i)$ for all examples to fit an RRT

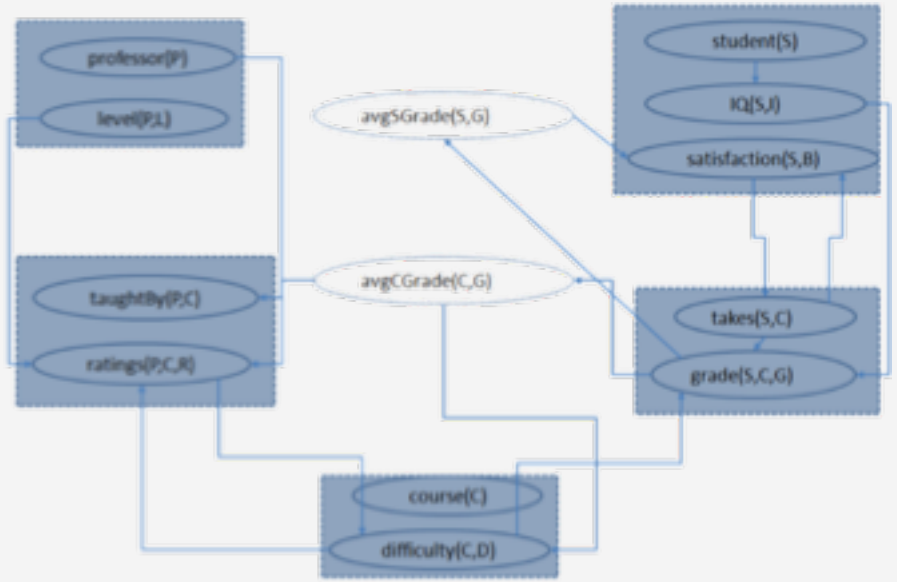
Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, Jude Shavlik: Gradient-based Boosting for Statistical Relational Learning: The Relational Dependency Network Case. In *Machine Learning*, 2012.

RDN-Boost: Overview

- For each $\psi\big(A_k; parents(A_k)\big) \in F$
  - 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

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





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** ▷ Iterate through K predicates
    for $1 \leq m \leq M$ **do** ▷ Iterate through M gradient steps
      $S_k \leftarrow \text{GenExamples}\left(k; Data; F_{m-1}^k\right)$ ▷ Generate examples
      $\Delta_m(k) \leftarrow \text{FitRelRegressTree}(S_k; L; D)$ ▷ Functional gradient
      $F_m^k \leftarrow F_{m-1}^k + \Delta_m(k)$ ▷ Update model
    $P(a_k|\text{Pa}(A_k)) \propto \psi^k$ ▷ ψ^k is obtained by grounding F_M^k

function GenExamples$(k, Data, F)$
  $S \leftarrow \emptyset$
  for $1 \leq i \leq N_k$ **do** ▷ Iterative over all examples
    Compute $P\left(y_k^i = 1; x_k^i\right), x_k^i = parents\left(y_k^i\right)$ ▷ Probability of y_k^i being true
    $\Delta\left(y_k^i; x_k^i\right) \leftarrow I\left(y_k^i = 1\right) - P\left(y_k^i = 1; x_k^i\right)$ ▷ Compute gradient
    $S \leftarrow S \cup \left\{\left(\left(x_k^i, y_k^i\right), \Delta\left(y_k^i; x_k^i\right)\right)\right\}$ ▷ Update regression examples
  return S

TreeBoostForRDNs

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.

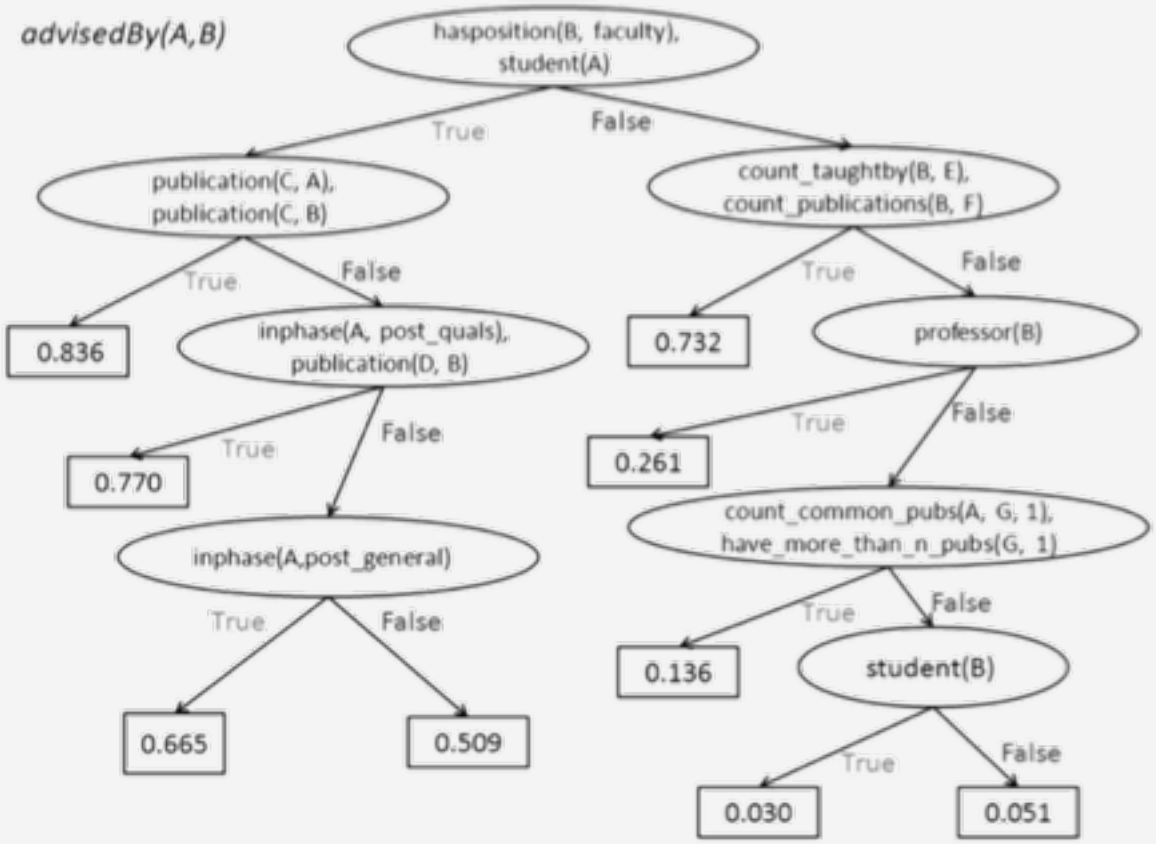
Bagging

- 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

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.

Interpretability

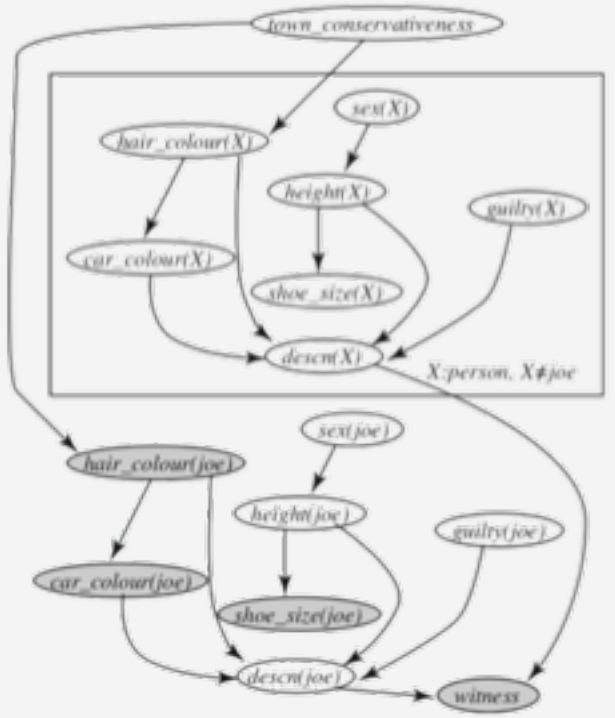
- Set of RRTs not really interpretable
  - Combine all RRTs into one tree and produce probabilities in leaves for interpretation by humans



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.

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 probabilistic relational models



Changing Domains

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

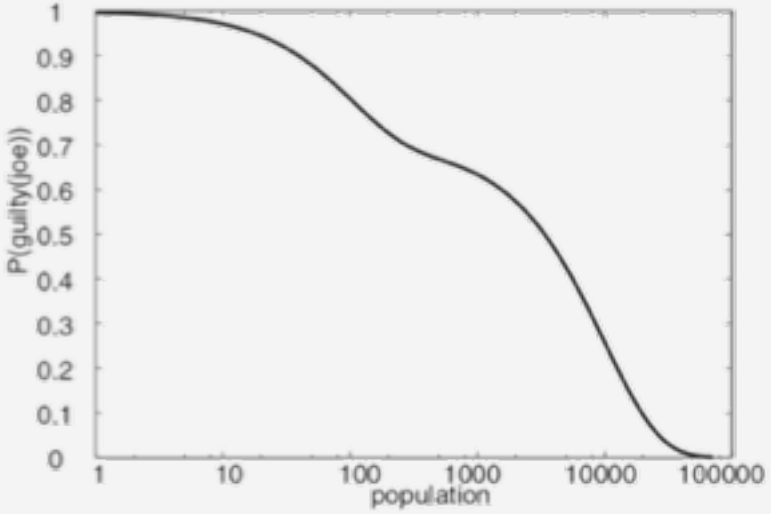
Known Universe

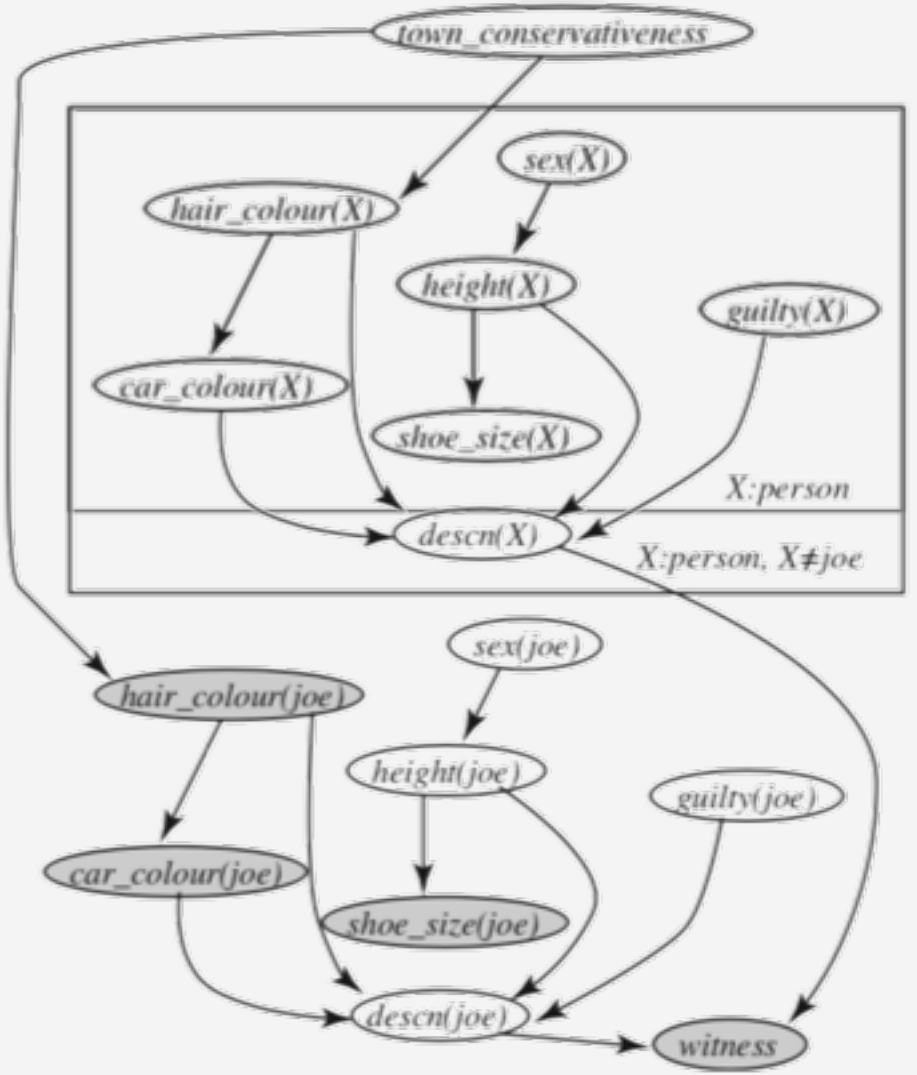
- Lifting based on knowing your universe
  - Universe = set of domains for logical variables in model
- Grounding semantics is only defined given specific domains for logical variables
  - Evidence for known constants
  - Queries reference known constants
- Potentials in parfactors learned with an underlying, 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 parfactors accurately describe world
- Posterior probabilities change depending on domain sizes
  - Example by Poole (2003)



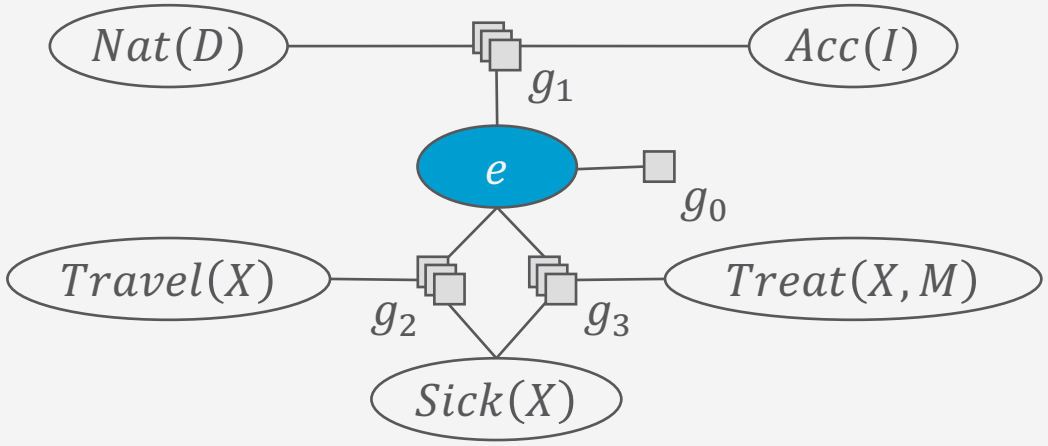


David Poole. First-order Probabilistic Inference. In *IJCAI-03 Proc. of the 18th Internat. Joint Conf. on AI*, 2003.

… Without Effects

- (Conditional) Independence
  PRVs, containing logical variables $\boldsymbol{X}$, that are (conditionally) independent from query terms
  $\rightarrow$ domains of $\boldsymbol{X}$ have no influence on query results
- E.g., given $Epid = e$,
  - $\mathrm{dom}(D)$ and $\mathrm{dom}(I)$ do not matter for queries regarding $Travel$, $Sick$, and $Treat$
  - $\mathrm{dom}(X)$ and $\mathrm{dom}(M)$ do not matter for queries regarding Nat and Man

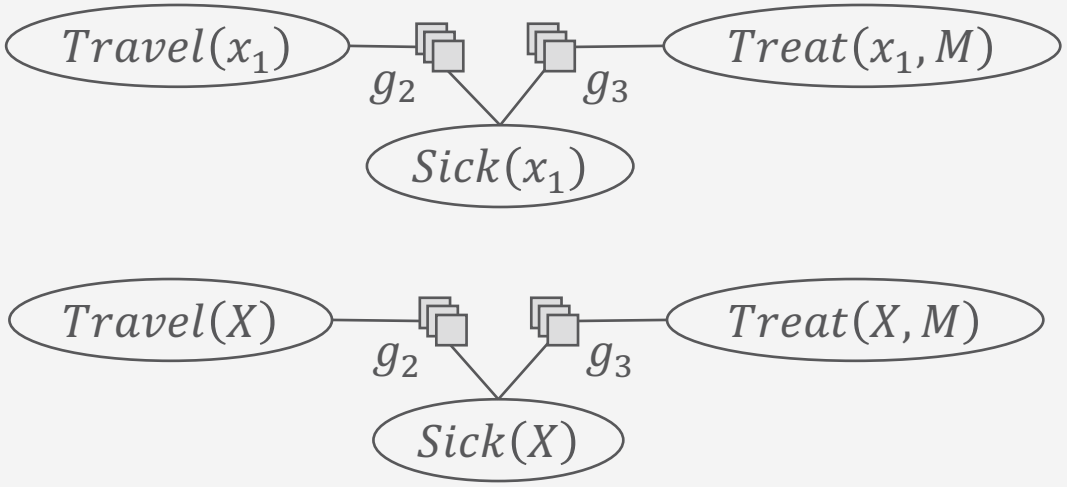
$\rightarrow$ Partly invariant under increasing domain sizes



… Without Effects

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

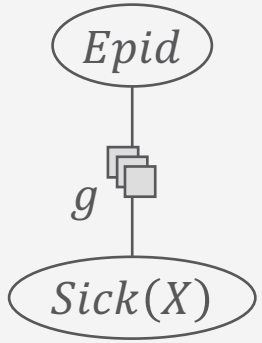
→ Partly invariant under increasing domain sizes



Manfred Jaeger and Oliver Schulte. Inference, Learning, and Population Size: Projectivity for SRL Models. In *StaRAI-18 Workshop on Statistical Relational Artificial Intelligence*, 2018.

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



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

$Epid$	g'
$false$	a
$true$	b

$Epid$	g''
$false$	a^n
$true$	b^n

$Epid$	g^n
$false$	$\frac{a^n}{a^n + b^n}$
$true$	$\frac{b^n}{a^n + b^n}$

$$\frac{1}{1 + \left(\frac{b}{a}\right)^n}$$

Sigmoid function

$n \to \infty$

$b > a$ 0

$a > b$ 1

David Poole, David Buchman, Seyed Mehran Kazemi, Kristian Kersting, and Sriraam Natarajan. Population Size Extrapolation in Relational Probabilistic Modeling. In *SUM-14 Proceedings of the 8th International Conference on Scalable Uncertainty Management*, 2014.

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

Happy Mittal, Ayush Bhardwaj, Vibhav Gogate, and Parag Singla. Domain-size Aware Markov Logic Networks. In *AISTATS-19 Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

Interim Summary*

- Changing domains while keeping the local distributions
  - Can lead to changes in posteriors
    - Extreme behaviour possible if domain size varies too much from original setting
      - More instances have to share the same probability mass
  - May have no consequence if
    - Evidence blocks any influence (through conditioning)
    - Shattering blocks any influence (by making parts independent)
    - Generalisation: projectivity
- Domain-size aware MLNs
  - To combat extreme behaviour under changing domain sizes
  - Adapt weights in MLNs based on the number of links between atoms when grounding
  - Analogous approach possible for parfactors

*Only done orally during lecture; slide added afterwards

Outline: 4. Lifted Learning

A. Overview of (propositional) learning
- Parameter and structure learning

B. Lifted encoding of propositional models
- Colour passing

C. Relation(al) learning
- First-order inductive learning, decision tree representation
- Relational dependency network learning
- Changing domains

D. Approximating Symmetries
- Evidence and model-based approaches, local structure

Approximating Symmetries - Motivation

- Most real world datasets do not induce symmetries beforehand
  - Lifted inference algorithms either…
    - not directly applicable or
    - resort to propositional inference for many random variables due to groundings
- Evidence breaks symmetries even in relational models
  - Marginal distributions of random variables distinguishable due to evidence
  - Not possible to treat them as indistinguishable objects during inference
- Detecting symmetries / constructing a symmetric model not advantageous
  - Cost for constructing a lifted model vs. actual benefits w.r.t efficiency
  - Size of lifted model may not be that much smaller than actual propositional model
- Hence: Efficient (approximate) inference through approximation of symmetries
  - Especially exploiting symmetries that almost hold

Evidence-based Approximation

- In general: Evidence may break symmetries
  - Major drawback in lifted inference algorithms
    - Many tasks involve the computation of conditional probabilities
  - Lifted inference not as efficient / beneficial as before due to groundings
    - With large domain sizes, grounding results in a huge model
- Two approximate approaches for efficiently handling evidence
  - Boolean Matrix Factorisation (BMF)
  - Domain Clustering

Evidence-based Approximation – Boolean Matrix Factorisation

- Consider a relational model, e.g., an MLN
  - evidence often given as a conjunction of literals (which represent ground atoms)
- It has been shown that conditioning on
  - Unary evidence can be tractable
    - Evidence on atoms consisting of predicates p_i/n of arity $n = 1$
      - E.g., $Sick(alice) \wedge \neg Sick(bob)$
    - Attributes of objects, i.e., no relations
  - Binary evidence is (in general) not tractable $(\#P\text{-}hard)$
    - Evidence on atoms consisting of predicates p_i/n of arity $n = 2$
      - E.g., $Treat(alice, m_1) \wedge \neg Treat(bob, m_2)$
    - Expresses relations among objects
- General idea: Encode binary relations through unary relations

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation – Boolean Matrix Factorisation

- Idea: Encode binary relations through unary relations
  - Represent unary evidence as a Boolean vector
    - Introduce vector for each predicate
    - Examples:
      - $q(X)$: $\neg q(x_1) \wedge q(x_2) \wedge \neg q(x_3) \wedge q(x_4)$
      - $r(Y)$: $\neg r(y_1) \wedge r(y_2)$
  - Represent binary evidence as a Boolean matrix
    - Example:
      - $\neg p(x_1, y_1) \wedge \neg p(x_1, y_2) \wedge p(x_2, y_1) \wedge \neg p(x_2, y_2) \wedge \neg p(x_3, y_1) \wedge p(x_3, y_2) \wedge p(x_4, y_1) \wedge p(x_4, y_2)$

$$q(X) \qquad r(Y)$$

$$\begin{matrix} X = x_1 \\ X = x_2 \\ X = x_3 \\ X = x_4 \end{matrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{matrix} Y = y_1 \\ Y = y_2 \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$p(X, Y) \qquad \begin{matrix} Y = y_1 & Y = y_2 \end{matrix}$$

$$\begin{matrix} X = x_1 \\ X = x_2 \\ X = x_3 \\ X = x_4 \end{matrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation – Boolean Matrix Factorisation

- Idea: Encode binary relations through unary relations
  - How to relate binary relations with unary relations?
    - Through formulas, e.g., $\forall X, \forall Y, p(X,Y) \Leftrightarrow q(X) \wedge r(Y)$
    - Conditioning on predicates q and r allows for conditioning on certain binary relations p
      - Indirect conditioning
  - Exploiting this formula by means of expressing the Boolean matrix for $p(X,Y)$ as an outer factorisation

Assuming evidence on predicates q and r, adding such formulas to the model does not change the probability distributions over atoms in the original model.

$$p(X,Y) \quad \begin{matrix} & Y=y_1 & Y=y_2 \\ X=x_1 & 0 & 0 \\ X=x_2 & 0 & 1 \\ X=x_3 & 0 & 0 \\ X=x_4 & 0 & 1 \end{matrix} = q(X) \begin{matrix} X=x_1 \\ X=x_2 \\ X=x_3 \\ X=x_4 \end{matrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \cdot r(Y) \begin{matrix} Y=y_1 \\ Y=y_2 \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T \qquad (\boldsymbol{P = qr^T})$$

Instead of conditioning on the 8 binary relation literals, it suffices to condition on the 6 unary relation literals

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation – Boolean Matrix Factorisation

- Idea: Encode binary relations through unary relations
  - Generalisation: Introduce multiple unary relations
    - E.g., $\forall X, \forall Y, p(X,Y) \Leftrightarrow \big(q_1(X) \wedge r_1(Y)\big) \vee \big(q_2(X) \wedge r_2(Y)\big) \vee \cdots \vee (q_n(X) \wedge r_n(Y))$
      - Relations which can be represented by the sum of outer products
    - $\boldsymbol{P} = \boldsymbol{q}_1\boldsymbol{r}_1^T \vee \boldsymbol{q}_2\boldsymbol{r}_2^T \vee \cdots \vee \boldsymbol{q}_n\boldsymbol{r}_n^T = \boldsymbol{Q}\boldsymbol{R}^T$
      - Sum: Treat $\vee$ as $+$ and $1 \vee 1 = 1$
      - Matrix multiplication in Boolean algebra, e.g., $(\boldsymbol{Q}\boldsymbol{R}^T)_{i,j} = \bigvee_r (\boldsymbol{Q}_{i,r} \wedge \boldsymbol{R}_{j,r})$

$$\boldsymbol{P} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}^T \vee \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}^T \vee \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}^T = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}}_{\boldsymbol{Q}} \cdot \underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}^T}_{\boldsymbol{R}}$$

Factorisation $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{R}^T$ called Boolean Matrix Factorisation (BMF)

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation – Boolean Matrix Factorisation

- Idea: Encode binary relations through unary relations

$$\underset{(4\times4)}{\boldsymbol{P}} = \begin{bmatrix}1&1&0&0\\1&1&0&1\\0&0&1&0\\1&0&0&1\end{bmatrix} = \begin{bmatrix}0\\1\\0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\\0\\1\end{bmatrix}^T \vee \begin{bmatrix}1\\1\\0\\0\end{bmatrix}\cdot\begin{bmatrix}1\\1\\0\\0\end{bmatrix}^T \vee \begin{bmatrix}0\\0\\1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\0\\1\\0\end{bmatrix}^T = \underset{(4\times3)}{\boldsymbol{Q}}\begin{bmatrix}0&1&0\\1&1&0\\0&0&1\\1&0&0\end{bmatrix}\cdot\underset{(4\times3)}{\boldsymbol{R}}\begin{bmatrix}1&1&0\\0&1&0\\0&0&1\\1&0&0\end{bmatrix}^T$$

Boolean rank $n = 3$

- BMF $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{R}^T$ factorises a $(k\times l)$ matrix $\boldsymbol{P}$ into
  - a $(k\times n)$ matrix $\boldsymbol{Q}$ and
  - a $(l\times n)$ matrix $\boldsymbol{R}$
  - with $n \leq \min(k, l)$

Potentially $n \ll k$ and $n \ll l$

- **Any** Boolean matrix can be expressed by a BMF
  - Boolean rank of $\boldsymbol{P}$: smallest n for which it is possible to factorise $\boldsymbol{P}$

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation – Boolean Matrix Factorisation

- Idea: Encode binary relations through unary relations

$$\underset{(4\times4)}{\boldsymbol{P} = \begin{bmatrix}1&1&0&0\\1&1&0&1\\0&0&1&0\\1&0&0&1\end{bmatrix}} = \begin{bmatrix}0\\1\\0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\\0\\1\end{bmatrix}^T \vee \begin{bmatrix}1\\1\\0\\0\end{bmatrix}\cdot\begin{bmatrix}1\\1\\0\\0\end{bmatrix}^T \vee \begin{bmatrix}0\\0\\1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\0\\1\\0\end{bmatrix}^T = \underset{(4\times3)}{\overset{\boldsymbol{Q}}{\begin{bmatrix}0&1&0\\1&1&0\\0&0&1\\1&0&0\end{bmatrix}}}\cdot\underset{(4\times3)}{\overset{\boldsymbol{R}}{\begin{bmatrix}1&1&0\\0&1&0\\0&0&1\\1&0&0\end{bmatrix}^T}}$$

- BMF $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{R}^T$
  - Problem: Many matrices do have nearly full rank
  - Reduce rank by means of approximation
    - Find $\boldsymbol{Q}$ and $\boldsymbol{R}$ s.t. $\boldsymbol{P} \approx \boldsymbol{Q}\boldsymbol{R}^T$
      - With a smaller Boolean rank
- Low-rank approximate BMF

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation – Boolean Matrix Factorisation

- Idea: Encode binary relations through unary relations

$$\boldsymbol{P} = \underset{(4\times4)}{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}^T \vee \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}^T \vee \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}^T = \underset{(4\times3)}{\overset{\boldsymbol{Q}}{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}}} \cdot \underset{(4\times3)}{\overset{\boldsymbol{R}}{\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}^T}}$$

approx.

$$\boldsymbol{P} \approx \underset{(4\times4)}{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}^T \vee \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}^T \underbrace{\vee \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}^T}_{\text{Not required anymore}} = \underset{(4\times2)}{\overset{\boldsymbol{Q}}{\begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}}} \cdot \underset{(4\times2)}{\overset{\boldsymbol{R}}{\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}^T}}$$

- Boolean rank of $\boldsymbol{P}$ is now 2 instead of 3
- Less unary literals required

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation – Boolean Matrix Factorisation

- Idea: Encode binary relations through unary relations
- General idea of low-rank approximate BMF:
  - instead of computing query $P(q \mid e)$ exactly, compute $P(q \mid e')$ approximately
    - I.e., approximate the binary evidence e by unary evidence e'
  - Obtain e' as follows:
    - Find an approximate BMF of the binary evidence represented as a Boolean matrix $\boldsymbol{P}$
    - Extract unary relations from the BMF of $\boldsymbol{P}$
    - Add formulas to MLN which relate binary literals to unary literals
    - Use unary literals to express evidence e
- BMF allows for parameterising the complexity of conditioning in lifted inference
- More details, especially on the complexity of this approach, can be found in the source below

Guy Van den Broeck and Adnan Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems*, 26, 2013.

Evidence-based Approximation

- In general: Evidence may break symmetries
  - Major drawback in lifted inference algorithms
    - Many tasks involve the computation of conditional probabilities
  - Lifted inference not as efficient / beneficial as before due to groundings
    - With large domain sizes, grounding results in a huge model
- Two approximate approaches for efficiently handling evidence:
  - Boolean Matrix Factorisation (BMF)
  - Domain Clustering

Evidence-based Approximation – Domain Clustering

- Another evidence-based approach: Domain Clustering
- For an MLN Ψ, the domain sizes determine the size of the resulting grounded model $G_\Psi = gr(\Psi)$
  - Each possible grounding of a formula has to be considered
- If Ψ induces few symmetries or symmetry-breaking evidence is present, (lifted) inference has to be performed on a possibly large model
- Idea: Reduce the total number of formulas by means of reducing the domain sizes
  - Results in a smaller MLN
  - Clustering approach in order to merge “similar” objects
    - Introduce distance function for measuring similarity of objects based on the evidence
    - Use representatives of clusters (cluster-centres) for inference

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain clustering based on evidence
- MLN Ψ with M predicates $p_1, \dots, p_M$ and N weighted formulas $\psi_1, \dots, \psi_N$
- Let $\boldsymbol{I}$ be a set of indices (i, j) with $1 \leq i \leq M, 1 \leq j \leq A_i$
  - A_i is the arity of the i-th predicate
  - (i, j) refers to the j-th argument of the i-th predicate in Ψ
- Define binary relation R s.t.
  - $(i, j)\ R\ (a, b) \Leftrightarrow$ there exists a formula $\psi \in \Psi$ with
    - ψ contains atoms having predicate symbols indexed by i and a
    - Logical variable X of ψ appears as the j-th argument and as the b-th argument of atoms having predicate symbols indexed by i and a respectively.
- R^+ is an equivalence relation on $\boldsymbol{I}$
  - R^+: transitive closure of R on $\boldsymbol{I}$
- $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_P\}$ denotes the set of equivalence classes of $\boldsymbol{I}$ w.r.t. R^+
  - Required in order to define the clustering problem

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain Clustering based on evidence
- Example: MLN Ψ with one formula $\psi = p_1(X, Y) \wedge p_2(Y, Z) \Rightarrow p_3(Z, X)$
  - $dom(X) = dom(Y) = dom(Z) = \{a, b\}$
  - $\mathcal{I} = \{\{(1,1),(3,2)\},\{(1,2),(2,1)\},\{(2,2),(3,1)\}\}$
  - $dom(\mathcal{I}_1) = dom(\{(1,1),(3,2)\}) = dom(X) = \{a, b\}$
  - Grounding $\mathcal{I}_1$ with a yields the (partially) ground formula $p_1(a, Y) \wedge p_2(Y, Z) \Rightarrow p_3(Z, a)$

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain Clustering based on evidence
- Approach: Reduce domain sizes and groundings of each $\mathcal{I}_k \in \mathcal{I}$ by introducing clusters
  - Group objects in each $\mathcal{I}_k$ to obtain clusters represented by cluster-centres (representatives)
  - Cluster-centre does not have to be an actual object in $dom(\mathcal{I}_k)$
- Formally,
  - Learn a new domain $\widehat{dom}\,(\mathcal{I}_k)$ and a surjective mapping $\zeta: dom(\mathcal{I}_k) \rightarrow \widehat{dom}\,(\mathcal{I}_k)$
    - $\widehat{dom}\,(\mathcal{I}_k) = \{\mu_{kj}\}_{j=1}^{r_k}$ with μ_{kj} the cluster-centre of the j-th cluster of $\mathcal{I}_k$ and r_k the number of clusters
    - Each object in $dom(\mathcal{I}_k)$ is assigned to a cluster(-centre) in $\widehat{dom}\,(\mathcal{I}_k)$ through this mapping
- Yields a new MLN $\widehat{\Psi}$ where each $dom(\mathcal{I}_k)$ is replaced by $\widehat{dom}\,(\mathcal{I}_k)$
  - New domain should be much smaller, $\left|\widehat{dom}\,(\mathcal{I}_k)\right| \ll |dom(\mathcal{I}_k)|$, as $\left|gr\left(\widehat{\Psi}\right)\right| < |gr(\Psi)|$

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain Clustering based on evidence
- Clustering problem defined as follows:

$$\min_{\boldsymbol{C}_1 \ldots \boldsymbol{C}_P} \sum_{k=1}^{P} \sum_{j=1}^{r_k} \sum_{C_{kj} \in \boldsymbol{C}_{kj}} d\left(C_{kj}, \mu_{kj}\right)$$

- With d as a distance measure between two groundings of $\mathcal{I}_k \in \mathcal{I}$ and r_k the number of clusters for $\mathcal{I}_k$
- $\boldsymbol{C}_{kj}$: all groundings of $\mathcal{I}_k$ in cluster j
- μ_{kj}: Cluster-centre of $\boldsymbol{C}_{kj}$ ($\zeta^{-1}\left(\mu_{kj}\right) = \boldsymbol{C}_{kj}$)
  - „Average grounding“ for that cluster

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain Clustering based on evidence
- Ground atoms in $\widehat{\Psi}$ and Ψ are different
  - Logical variables in $\widehat{\Psi}$ are substituted by cluster-centres rather than objects
  - One ground atom in $\widehat{\Psi}$ corresponds to multiple ground atoms in Ψ
- For inference, the evidence $\boldsymbol{e}$ needs to be modified since it consists of truth values assigned to ground atoms in Ψ
- Therefore: Approximation $\hat{\boldsymbol{e}}$ of evidence $\boldsymbol{e}$ required
  - $\hat{\boldsymbol{e}}$ consists of ground atoms in $\widehat{\Psi}$ (atoms grounded with cluster-centres)
  - Connection between $\boldsymbol{e}$ and $\hat{\boldsymbol{e}}$: a single ground atom in $\hat{\boldsymbol{e}}$ represents multiple ground atoms in $\boldsymbol{e}$
    - Expansion of a ground atom in $\widehat{\Psi}$: set of all groundings which are covered by this ground atom w.r.t the clustering
    - Ground atom in $\hat{\boldsymbol{e}}$ set to $true/false$: Every grounding in its expansion set to $true$ / $false$

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain Clustering based on evidence
- Given evidence $\boldsymbol{e}$, how to choose the truth values for the corresponding ground atoms in $\hat{\boldsymbol{e}}$?
- $\hat{\boldsymbol{e}}$ can be optimally chosen as follows:
  - With π_{ij} as the expansion of a grounding $\hat{e}$ in $\hat{\boldsymbol{e}}$ count in $\pi_{ij} \cap \boldsymbol{e}$
    - The number of positive-sign ($true$) elements (n_+)
    - The number of negative-sign ($false$) elements (n_-)
  - Set $\hat{e}$ to $true$ if $n_+ \geq \frac{|\pi_{ij}|}{2}$ and to $false$ if $n_- \geq \frac{|\pi_{ij}|}{2}$

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain Clustering based on evidence
- How to choose the clusters and the distance measure?
  - Clusters: Use any existing clustering algorithm (e.g., K-Means)
    - Clustering algorithm requires distance measure in order to cluster objects based on their similarity
  - Distance measure: Choose distance measure based on given evidence $\boldsymbol{e}$:

$$d\left(C_{kj_1}, C_{kj_2}\right) = ||\boldsymbol{U}_{C_{kj_1}} - \boldsymbol{U}_{C_{kj_2}}||$$

    - $\boldsymbol{U}_{C_{kj}} = c_{\psi_1}, \ldots, c_{\psi_N}$ feature vector
    - c_{ψ_k}: number of groundings in formula ψ_k of MLN $\Psi_{C_{kj}}$ satisfied due to evidence $\boldsymbol{e}$
    - $\Psi_{C_{kj}}$: MLN obtained after grounding $\mathcal{J}_k$ with the j-th constant in $dom(\mathcal{J}_k)$
    - "How close are the resulting MLNs after grounding with specific objects"
- More details on that can be found in the source below

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Evidence-based Approximation – Domain Clustering

- Idea: Domain Clustering based on evidence
- Algorithm proceeds as follows for an MLN Ψ:
  - Requirements: Clustering algorithm, distance function and inference algorithm specified
  1. Compute partition $\mathcal{J}$
  2. Apply the clustering algorithm to each $\mathcal{J}_k \in \mathcal{J}$
  3. Approximate $\widehat{dom}(\mathcal{J}_k)$ for each $dom(\mathcal{J}_k)$ to get $\widehat{\Psi}$
  4. Approximate evidence $\boldsymbol{e}$ based on reduced domains to get $\hat{\boldsymbol{e}}$
  5. Perform inference on $\widehat{\Psi}$ using the specified inference algorithm and $\hat{\boldsymbol{e}}$

Deepak Venugopal and Vibhav Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 2014.

Model-based Approximation

- Idea: Lift the model approximately
- Useful if
  - Constructing the lifted model computationally expensive
    - Includes identifying sets of indistinguishable variables
  - The resulting lifted model is close to the propositional model w.r.t model size
    - I.e., lifting yields little to no speedup
- Idea: group variables that behave similarly
  - Do not completely behave identical
  - Results potentially in a much smaller model
  - Reduces cost for lifting the model

Parag Singla, Aniruddh Nath, and Pedro Domingos: Approximate Lifting Techniques for Belief Propagation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2014.

Model-based Approximation

- Early Stopping introduced in the context of Lifted Belief Propagation (LBP)
  - LBP includes a step called lifted network construction (LNC)
    - Similar to the colour-passing algorithm, LNC identifies messages which are sent and received from multiple nodes by simulating BP
    - Constructs a model consisting of supernodes and superfeatures
      - Supernode: Set of atoms that send and receive exactly the same message throughout BP
      - Superfeature: Set of ground clauses that send and receive the same messages
  - Idea: Run LNC only up to a fixed number of iterations
    - Stop construction of lifted network after k iterations
    - Potentially faster than running to convergence
    - Approximate symmetries, but possible to bound the induced error

Parag Singla, Aniruddh Nath, and Pedro Domingos: Approximate Lifting Techniques for Belief Propagation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2014.

Application of Approximate Symmetries

- Use especially for asymmetric models
  - Allows for
    - Possibly more efficient (lifted) inference
    - Applying lifted inference techniques in asymmetric models
  - Models can be asymmetric by default or evidence breaks symmetries
  - But: query results, i.e., (conditional) probabilities may be biased
- Another application of approximate symmetries: Approximate Inference (sampling)
  - Lifted MH (briefly introduced in the previous chapter) incorporates approximate symmetries to construct an orbital Metropolis chain
    - Operates on approximately symmetric states

Guy Van den Broeck and Mathias Niepert: Lifted Probabilistic Inference for Asymmetric Graphical Models . In AAAI-15 Proceedings of the 29th AAAI Conference on Artificial Intelligence, 2015.

Local Structure

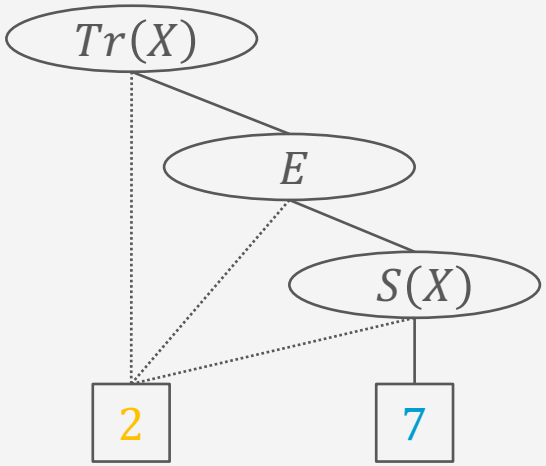
- What if approximation of (global) symmetries not suitable?
- One could consider local structures
  - Often referred to as local symmetries
  - Potential functions can have local symmetries which can be exploited
    - E.g., Potentials that a potential function maps to multiple times
- There are representations and approaches for incorporating local structures during inference
  - E.g., Algebraic Decision Diagrams (ADDs)

$Travel(X)$	$Epid$	$Sick(X)$	ϕ
$false$	$false$	$false$	2
$false$	$false$	$true$	2
$false$	$true$	$false$	2
$false$	$true$	$true$	2
$true$	$false$	$false$	2
$true$	$false$	$true$	2
$true$	$true$	$false$	2
$true$	$true$	$true$	7

$$(\ln 2\,, \neg travel(X) \vee \neg epid \vee \neg sick(X))$$
$$(\ln 7\,, travel(X) \wedge epid \wedge sick(X))$$

Local Structure – ADDs

- Represent function as a binary tree
  - Potentials are the leaves
  - Left / right child:
    Assignment of $false$ / $true$
- Requires Boolean range values
  - Possible to adapt to multi-valued ranges
- Multiplication and addition of factors implementable on ADDs
  - If ADD much smaller than table, fewer arithmetic operations to carry out overall
- But: Overhead in handling ADDs
  - Building / updating tree structure



$Travel(X)$	$Epid$	$Sick(X)$	ϕ
$false$	$false$	$false$	2
$false$	$false$	$true$	2
$false$	$true$	$false$	2
$false$	$true$	$true$	2
$true$	$false$	$false$	2
$true$	$false$	$true$	2
$true$	$true$	$false$	2
$true$	$true$	$true$	7

$$(\ln 2\,, \neg travel(X) \vee \neg epid \vee \neg sick(X))$$
$$(\ln 7\,, travel(X) \wedge epid \wedge sick(X))$$

R. Iris Bahar, E.A. Frohm, Charles M. Gaona, G.D. Hachtel, E. Macii, Abelardo Pardo, and F. Somenzi: Algebraic Decision Diagrams and Their Application. In: *IEEE /ACM International Conference on CAD*, 1993.
Mark Chavira and Adnan Darwiche: Compiling Bayesian Networks Using Variable Elimination. In: *IJCAI-07 Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007. (and in the book "Modelling and Reasoning with Bayesian Networks" by Adnan Darwiche, mentioned in Introduction; Chapter 13.5)

Interim Summary

- Approximating symmetries in order to exploit symmetries that “almost” hold
  - Asymmetric models, symmetry-breaking evidence, complexity of lifting a model
- Evidence and model-based approaches
  - Boolean Matrix Factorisation – express binary evidence through unary evidence
  - Domain Clustering – evidence-based clustering through cluster-centres
  - Approximate lifted model by means of, e.g., early stopping
- Local Structure
  - Capturing and exploiting local symmetries (e.g., ADDs)

Outline: 4. Lifted Learning

A. *Overview of propositional learning*
   - Parameter and structure learning

B. *Lifted encoding of propositional models*
   - Colour passing

C. *Relation(al) learning*
   - First-order inductive learning, decision tree representation
   - Relational dependency network learning
   - Changing domains

D. *Approximating Symmetries*
   - Evidence and model-based approaches, local structure

⇒ Next: Lifted Sequential Models and Inference

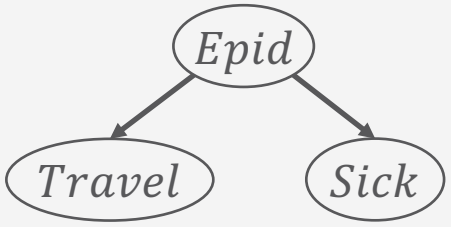
Appendix

ML-based Parameter Learning

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

$Travel$	$Epid$	$Sick$
$false$	$false$	$false$
$false$	$false$	$true$
$true$	$false$	$false$
$false$	$true$	$true$
$\vdots$	$\vdots$	$\vdots$

→ Probability distributions to learn

$P(Epid), P(Travel \mid Epid), P(Sick \mid Epid)$

$Epid$	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

Maximum-Likelihood Parameter Estimation

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

- Maximum Likelihood Estimation (MLE) principle: Choose $\boldsymbol{\theta}$ such that observed data (training data) most probable:

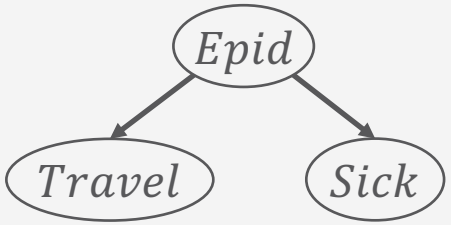
$$P(\boldsymbol{d} \mid \boldsymbol{\theta}) = \prod_{j=1}^{N} P(d_j \mid \boldsymbol{\theta})$$

i.i.d. data points

→ Probability distributions to learn

$P(Epid), P(Travel \mid Epid), P(Sick \mid Epid)$

Model



Data

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

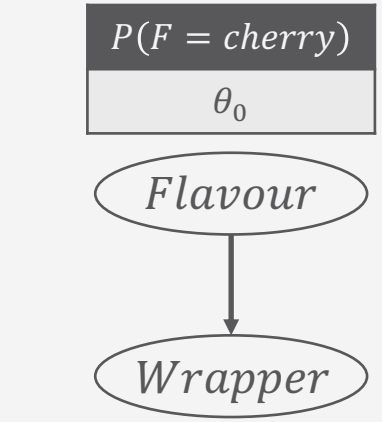
Epid	P
false	$1-\theta_0$
true	θ_0

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
    - $\theta_0, \theta_1, \theta_2$



F	$P(W = red \mid F)$
$cherry$	θ_1
$lime$	θ_2

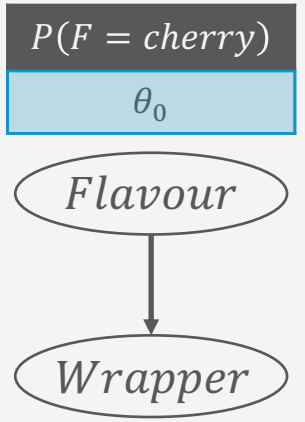
An Even Smaller Example

- Probability for, e.g., $W = yellow, F = cherry$

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

- 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:

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



F	$P(W = red \mid F)$
$cherry$	θ_1
$lime$	θ_2

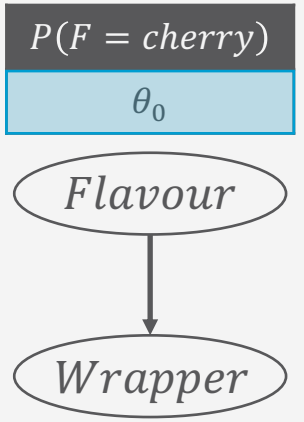
An Even Smaller Example

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

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

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

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



F	$P(W = red \mid F)$
cherry	θ_1
lime	θ_2

An Even Smaller Example

- Take derivatives w.r.t. $\boldsymbol{\theta}$ and determine maximal values by setting derivatives equal to 0
  - In the example:
    - $L(d \mid h_{\theta_0\theta_1\theta_2}) = c \log \theta_0 + l \log(1-\theta_0) + c_r \log \theta_1 + c_y \log(1-\theta_1) + l_r \log \theta_2 + l_y \log(1-\theta_2)$
    - Derivative w.r.t. θ_0

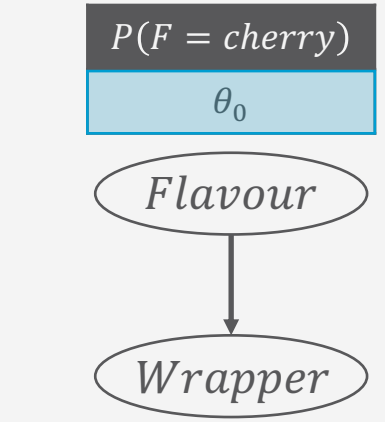
$$\frac{\partial L}{\partial \theta_0} = \frac{c}{\theta_0} - \frac{l}{1-\theta_0} = 0 \quad \Rightarrow \quad \theta_0 = \frac{c}{c+l}$$

    - 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

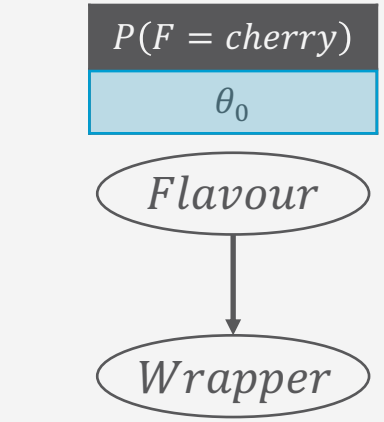
$$\frac{\partial L}{\partial \theta_2} = \frac{l_r}{\theta_2} - \frac{l_y}{1-\theta_2} = 0 \quad \Rightarrow \quad \theta_2 = \frac{l_r}{l_r+l_y}$$



F	$P(W = red \mid F)$
$cherry$	θ_1
$lime$	θ_2

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 $\theta = P(v|\mathrm{pa}(V))$ in the CPT of a node V given by frequency of the considered value v given the relevant parent values $\mathrm{pa}(V)$



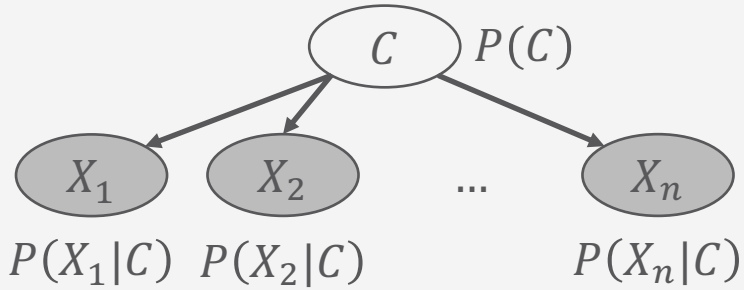
F	$P(W = red \mid F)$
$cherry$	θ_1
$lime$	θ_2

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 $P(\boldsymbol{d}|\boldsymbol{\theta})$, i.e., $h_{ML} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{d}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{j=1}^{N} P(d_j|\boldsymbol{\theta})$
  - Procedure
    1. Describe probabilities of data as a function of the parameters to be learning ($P(\boldsymbol{d} \mid \boldsymbol{\theta})$)
    2. Differentiate the log-likelihood ($\log P(\boldsymbol{d} \mid \boldsymbol{\theta})$) 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

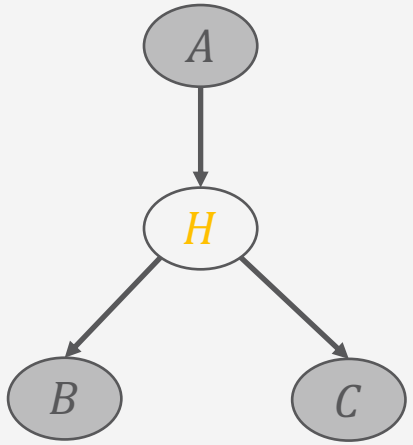


- Advantage: Relatively few parameters (entries in the CPTs) to learn
  - $2n + 1$ parameters for Boolean variables (general: $r(r-1)n + r - 1$)
    - $r = |\mathrm{Val}(C)|$, $r - 1$ entries in $P(C)$, $r(r-1)$ entries in $P(X_i|C)$ (for each c, learn $(r-1)$ entries)
- Form of *supervised learning*;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

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

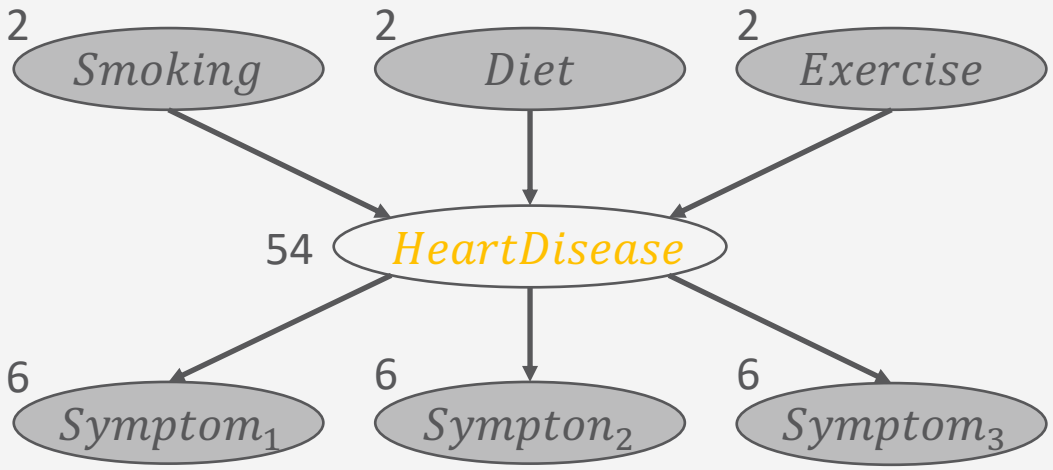
A	B	C
$true$	$false$	$true$
$false$	$true$	$true$
$true$	$true$	$false$
	$\vdots$	



Quick Fix

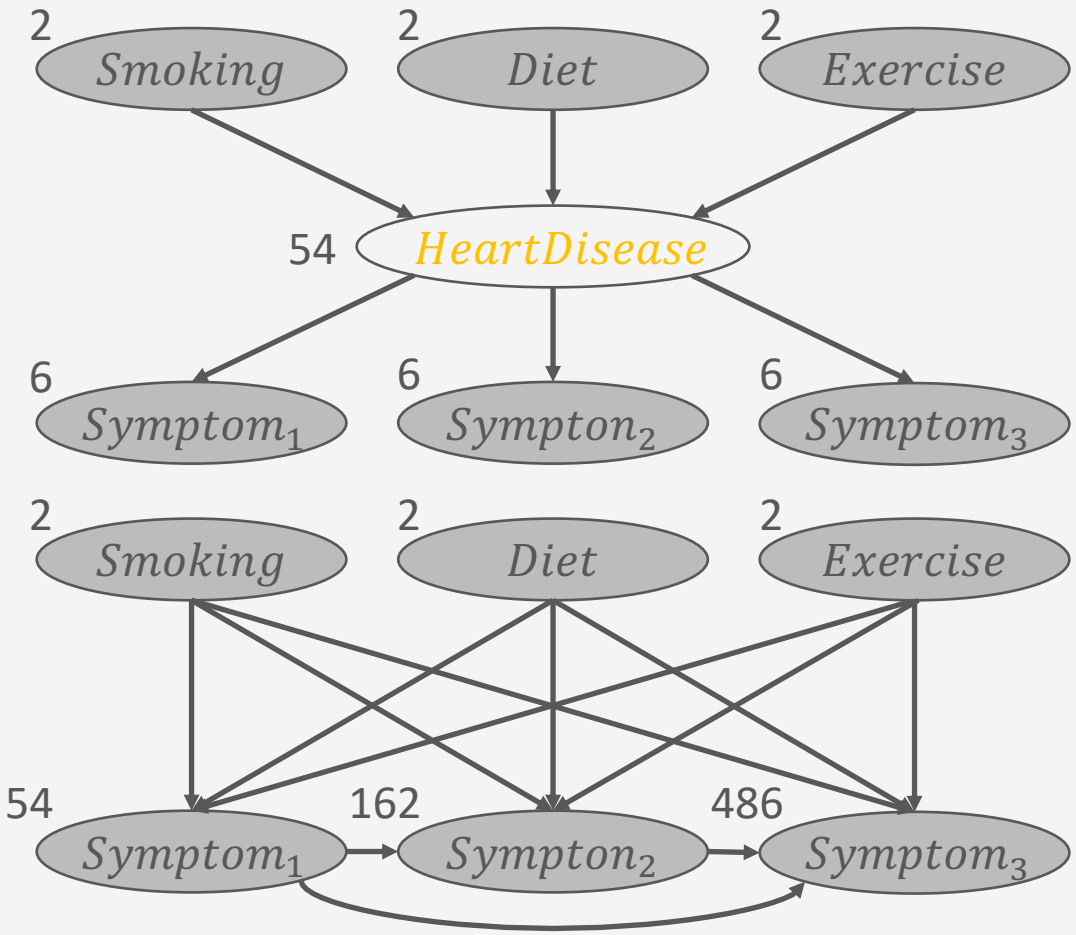
- Get rid of or avoid hidden variables
- May work in simple networks

- 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 → 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 infromation on the true nature of each data point

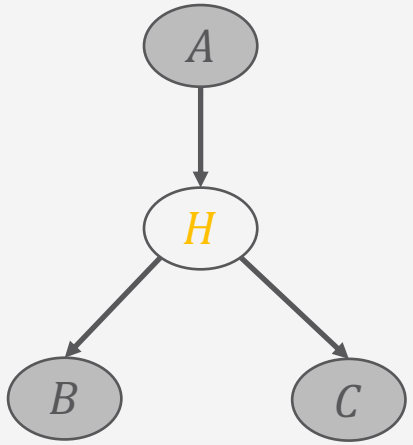
- Expectation-Maximisation (EM)
  - General algorithm for learning model parameters from incomplete data
  - Here, how it works on learning parameters for BNs with discrete variables

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
$true$	$false$	$true$
$false$	$true$	$true$
$true$	$true$	$false$
	$\vdots$	



Dempster, A.P., Laird. N.M., Rubin, D.B.: Maximum-Likelihood from Incomplete Data via the EM Algorithm. In: *Journal of the Royal Statistical Society*, **1977**

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)

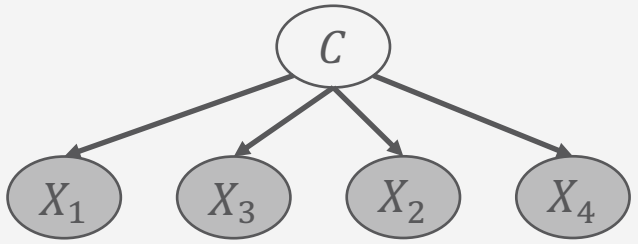
- It then refines this initial guess by cycling through two basic steps
  - Expectation (E):
    Update the data with predictions generated via the current model (expected counts)
  - Maximisation (M):
    Given the updated data, update the model parameters using MLE
    - Same step as when learning 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, $\mathrm{Val}(C) = \{1,2,3\}$

- Naïve Bayes classifier with hidden variable C

Model



→ probability distributions to learn

$P(C)$?
$P(X_1 \mid C)$?
$P(X_2 \mid C)$?
$P(X_3 \mid C)$?
$P(X_4 \mid C)$?

Data

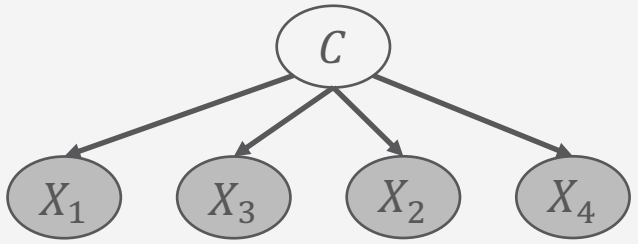
X_1	X_2	X_3	X_4
$true$	$false$	$true$	$true$
$false$	$true$	$true$	$false$
$true$	$false$	$true$	$true$
⋮			

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)

Model



→ probability distributions to learn

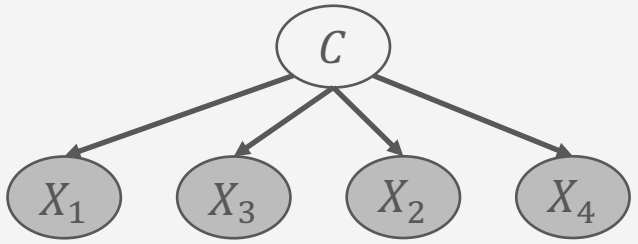
$P(C)$?
$P(X_1 \mid C)$?
$P(X_2 \mid C)$?
$P(X_3 \mid C)$?
$P(X_4 \mid C)$?

Define random parameters

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

- What would we need to learn parameters using MLE?
  - $P(C = k) = \frac{\#(datapoints\ with\ C{=}k)}{\#(all\ datapoints)}$ for each $k = 1,2,3$
  - $P(X_i = x \mid C = k) = \frac{\#(datapoints\ with\ X_i{=}x\ and\ C{=}k)}{\#(datapoints\ with\ C{=}k)}$ for each $x \in \mathrm{ran}(X_i), i = 1, \dots, 4, k = 1,2,3$
  - But, so far, we only have: $N = \#(all\ datapoints)$
    → 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

$P(C)$?
$P(X_1 \mid C)$?
$P(X_2 \mid C)$?
$P(X_3 \mid C)$?
$P(X_4 \mid C)$?

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

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

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

- Estimation then is:

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

- How do we get the necessary probabilities for $\widehat{N}$?
  - In Naïve Bayes model easy to compute:

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

Use 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

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

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

$$\widehat{N}(X_1 = true, C = 1) = \sum_{\boldsymbol{e}_j = (x_1, x_2, x_3, x_4), x_1 = true} P\left(C = 1 \middle| x_1^j = true, x_2^j, x_3^j, x_4^j\right)$$

Also computed using invented parameters

- Estimate $P(X_i \mid C)$

$$P(X_i = x \mid C = k) = \frac{exp\#(datapoints\ with\ X_i = x, C = k)}{exp\#(datapoints\ with\ C = k)} = \frac{\widehat{N}(X_i = x, C = k)}{\widehat{N}(C = k)}$$

  - for ach $x \in \mathrm{ran}(X_i), i = 1, \ldots, 4, k = 1,2,3$

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 paramteters by applying ML using the expected counts

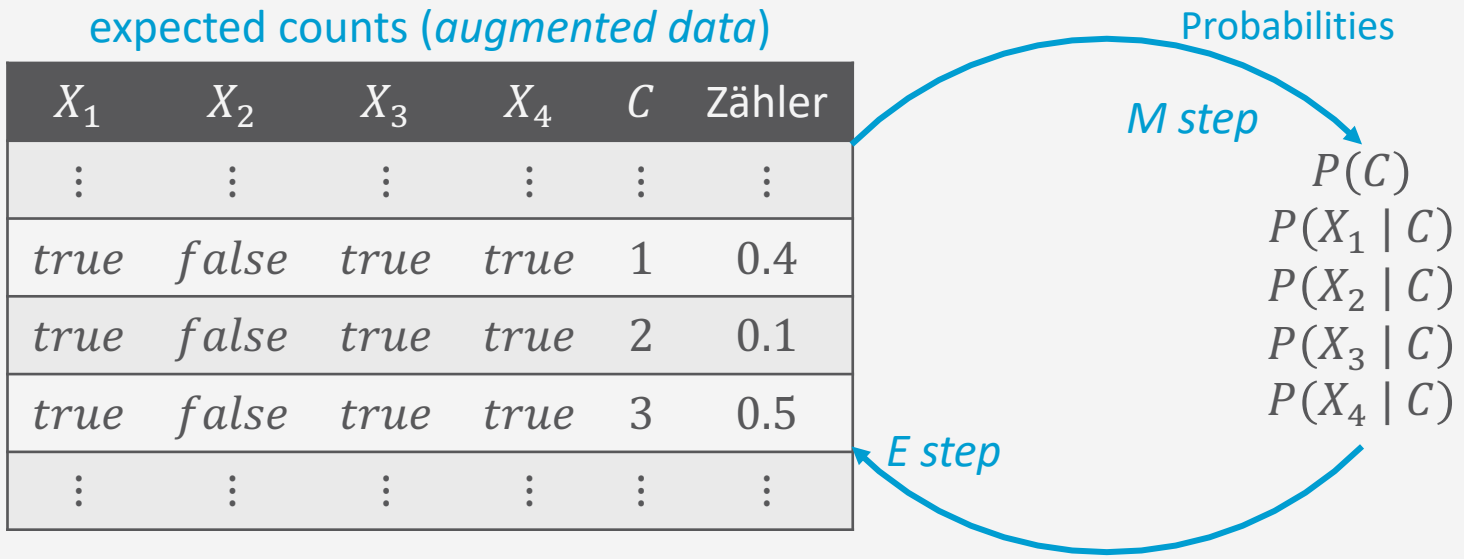
$$P(C = k) = \frac{\widehat{N}(C = k)}{N}$$

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

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

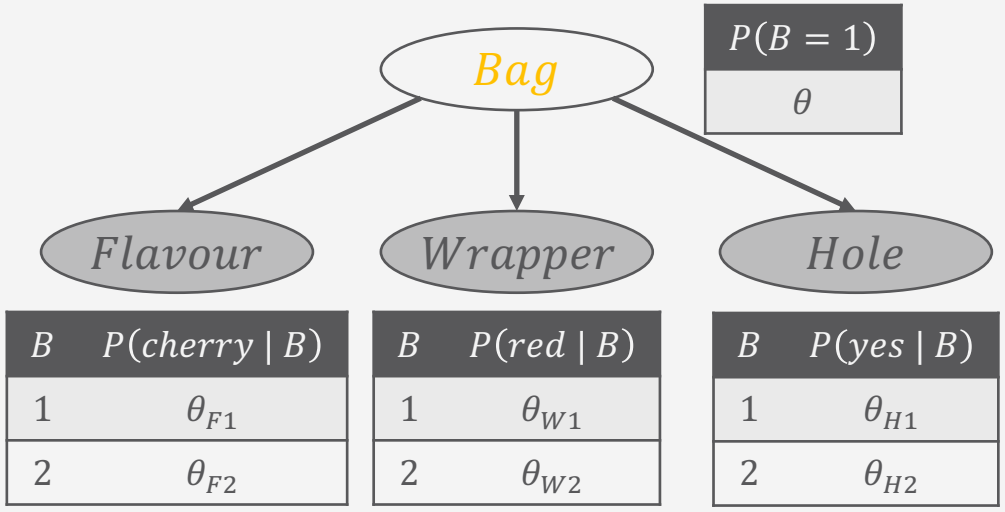
EM Cycle

- 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



B	$P(cherry \mid B)$
1	θ_{F1}
2	θ_{F2}

B	$P(red \mid B)$
1	θ_{W1}
2	θ_{W2}

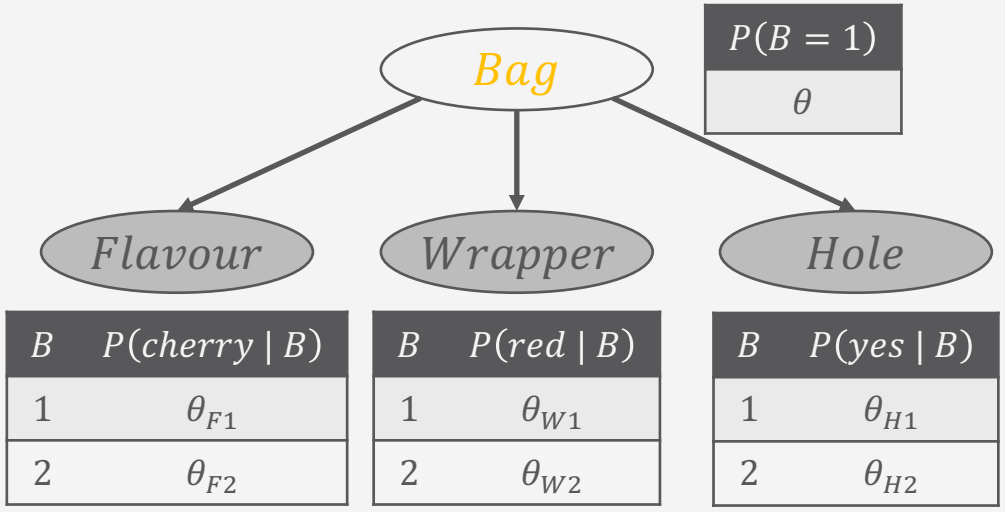
B	$P(yes \mid B)$
1	θ_{H1}
2	θ_{H2}

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



$P(B = 1)$
θ

B	$P(cherry \mid B)$
1	θ_{F1}
2	θ_{F2}

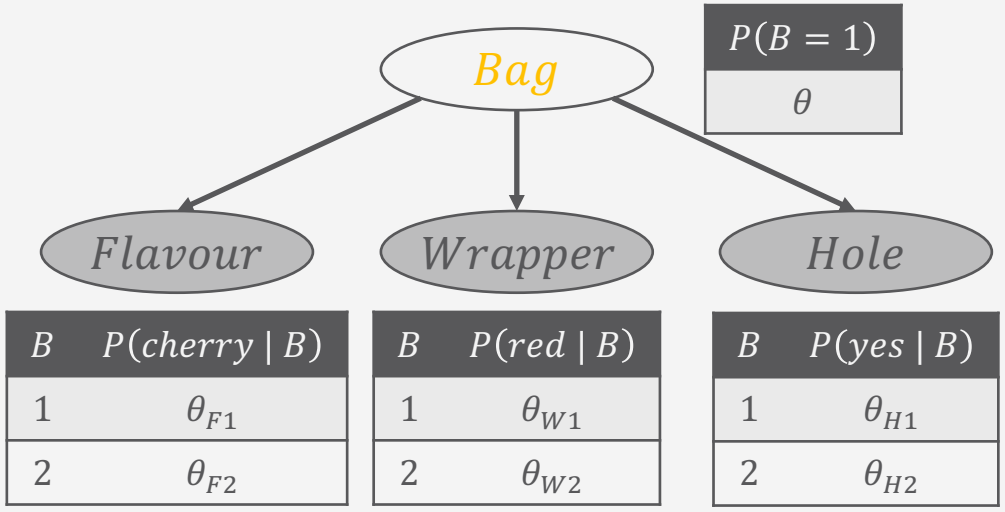
B	$P(red \mid B)$
1	θ_{W1}
2	θ_{W2}

B	$P(yes \mid B)$
1	θ_{H1}
2	θ_{H2}

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

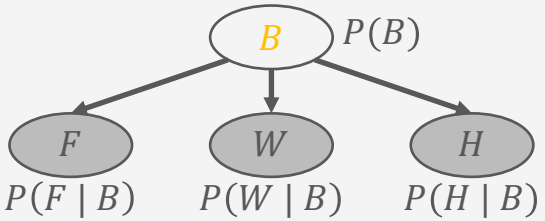


B	$P(cherry \mid B)$
1	θ_{F1}
2	θ_{F2}

B	$P(red \mid B)$
1	θ_{W1}
2	θ_{W2}

B	$P(yes \mid B)$
1	θ_{H1}
2	θ_{H2}

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

$$\widehat{N}(Bag = 1) = \sum_{j=1}^{N} P\big(Bag = 1 \mid f_j, w_j, h_j\big)$$

$$= \sum_{j=1}^{N} \frac{P\big(f_j, w_j, h_j \mid Bag = 1\big)P(Bag = 1)}{P\big(f_j, w_j, h_j\big)}$$

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

E Step

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

- 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

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

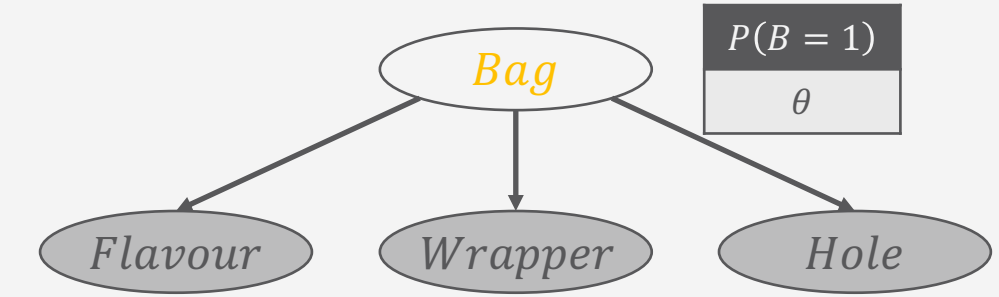
$$= 273 \frac{0.6^4}{0.6^4 + 0.4^4} = 273 \frac{0.1296}{0.1552} = 227.97$$

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

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

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

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



B	$P(cherry \mid B)$
1	θ_{F1}
2	θ_{F2}

B	$P(red \mid B)$
1	θ_{W1}
2	θ_{W2}

B	$P(yes \mid B)$
1	θ_{H1}
2	θ_{H2}

M Step

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

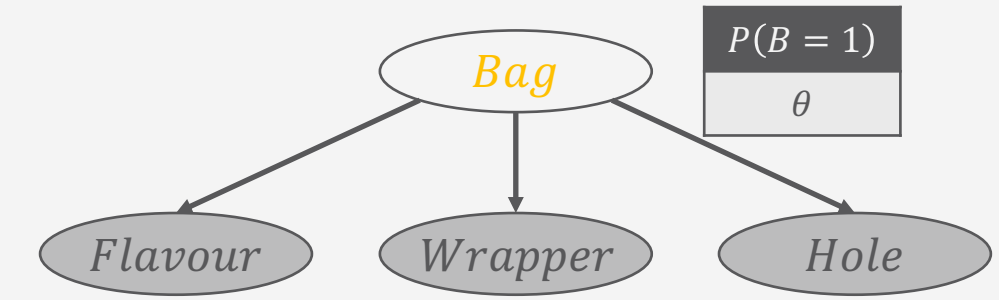
$$\widehat{N}(Bag = 1) = 612.4$$

- 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

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

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

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



B	$P(cherry \mid B)$
1	θ_{F1}
2	θ_{F2}

B	$P(red \mid B)$
1	θ_{W1}
2	θ_{W2}

B	$P(yes \mid B)$
1	θ_{H1}
2	θ_{H2}

Another Parameter: θ_{F1}

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

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

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

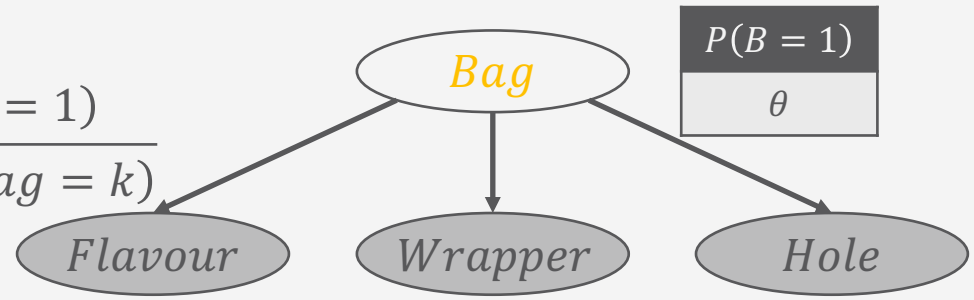
$$\hat{N}(B = 1, F = cherry) = \sum_{j:F=cherry} P\left(B = 1 \mid cherry_j, w_j, h_j\right)$$

  - Same as before:

$$\hat{N}(Bag = 1, F = cherry)$$
$$= \sum_{j=1}^{N} \frac{P\left(cherry_j \mid Bag = 1\right)P\left(w_j \mid Bag = 1\right)P\left(h_j \mid Bag = 1\right)P(Bag = 1)}{\sum_{k=1}^{2} P\left(cherry_j \mid Bag = k\right)P\left(w_j \mid Bag = k\right)P\left(h_j \mid Bag = k\right)P(Bag = k)}$$

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

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



$P(B = 1)$
θ

B	$P(cherry \mid B)$
1	θ_{F1}
2	θ_{F2}

B	$P(red \mid B)$
1	θ_{W1}
2	θ_{W2}

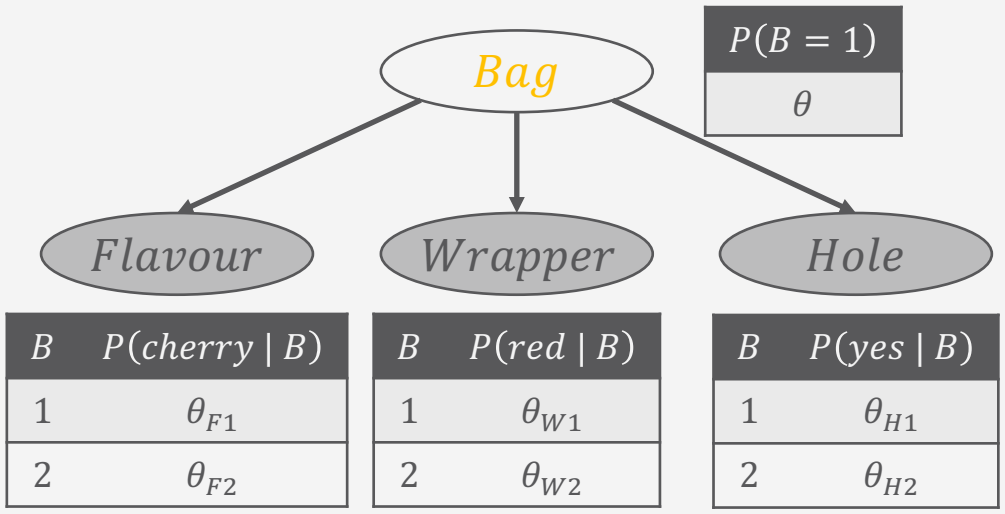
B	$P(yes \mid B)$
1	θ_{H1}
2	θ_{H2}

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



B	$P(cherry \mid B)$
1	θ_{F1}
2	θ_{F2}

B	$P(red \mid B)$
1	θ_{W1}
2	θ_{W2}

B	$P(yes \mid B)$
1	θ_{H1}
2	θ_{H2}

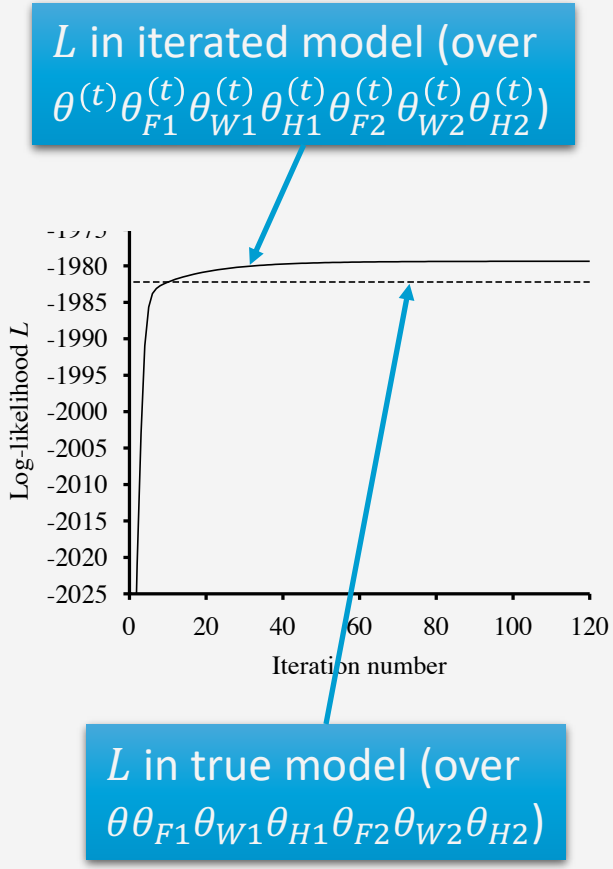
Learning Result

- Log likelihood of parameters:

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

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

  - One can show that this value increases with each EM cycle (convergence)
    - Example candy factory: Log likelihood $L = \log P(d, h_{\boldsymbol{\theta}})$ 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 $\mathrm{pa}(R_i)$

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

$$\theta_{ijk} = \frac{\widehat{N}\big(R_i = r_{ij}, \mathrm{Pa}(R_i) = \mathrm{pa}_k\big)}{\widehat{N}(\mathrm{Pa}(R_i) = \mathrm{pa}_k)}$$

  - Expected counts computed by summing over samples, after having computed all necessary $P\big(R_i = r_{ij}, \mathrm{Pa}(R_i) = \mathrm{pa}_{ik}\big)$ 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

$$P_F = \frac{1}{Z}\prod_{i=1}^{n}\phi_i(R_1, \dots, R_k)$$

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

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

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

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

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

   and take the logarithm

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

ML Procedure for Undirected Models

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

$\phi_f(\boldsymbol{r}_f)$ refers to parameter $\in \theta$

$$= \sum_{\boldsymbol{r} \in W} \#(\boldsymbol{r}) \log P(\boldsymbol{r}|\theta) = \sum_{\boldsymbol{r} \in W} \#(\boldsymbol{r}) \log \left(\frac{1}{Z} \prod_{f \in F} \phi_f(\boldsymbol{r}_f) \right)$$

$$= \sum_{\boldsymbol{r} \in W} \#(\boldsymbol{r}) \left(\log \left(\frac{1}{Z} \right) + \log \prod_{f \in F} \phi_f(\boldsymbol{r}_f) \right)$$

$$= \sum_{\boldsymbol{r} \in W} \#(\boldsymbol{r}) \left(-\log(Z) + \sum_{f \in F} \log \phi_f(\boldsymbol{r}_f) \right)$$

$$= \left(\sum_{\boldsymbol{r} \in W} \#(\boldsymbol{r}) \sum_{f \in F} \log \phi_f(\boldsymbol{r}_f) \right) - \sum_{\boldsymbol{r} \in W} \#(\boldsymbol{r}) \log(Z)$$

$$= \left(\sum_{\boldsymbol{r} \in W} \#(\boldsymbol{r}) \sum_{f \in F} \log \phi_f(\boldsymbol{r}_f) \right) - k \log(Z)$$

$$= \left(\sum_{f \in F} \sum_{\boldsymbol{r}_f \in \mathcal{R}(rv(f))} \#(\boldsymbol{r}_f) \log \phi_f(\boldsymbol{r}_f) \right) - k \log(Z)$$

ML Procedure for Undirected Models

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

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

- Derivation of l_1

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

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

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 $l_2 = k \log(Z)$

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

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

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

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

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

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

ML Procedure for Undirected Models

- Derivation of l_2

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

$$= k \cdot \frac{1}{\phi_f(\boldsymbol{r}_f)} \cdot \sum_{\boldsymbol{s} \in W} \delta(\boldsymbol{r}_f, \boldsymbol{s}_f) \cdot \frac{1}{Z} \cdot \prod_{e \in F} \phi_e(\boldsymbol{s}_e)$$

$$= k \cdot \frac{1}{\phi_f(\boldsymbol{r}_f)} \cdot \sum_{\boldsymbol{s} \in W} \delta(\boldsymbol{r}_f, \boldsymbol{s}_f) \cdot P(\boldsymbol{s})$$

$$= k \cdot \frac{1}{\phi_f(\boldsymbol{r}_f)} \cdot P(\boldsymbol{r}_f)$$

ML Procedure for Undirected Models

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

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

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

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

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

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

ML Procedure for Undirected Models

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

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

$$\#(\boldsymbol{r}_f) = k \cdot P(\boldsymbol{r}_f)$$

$$\frac{\#(\boldsymbol{r}_f)}{k} = P(\boldsymbol{r}_f)$$

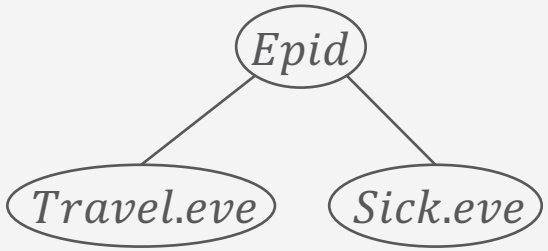
- States that the ML estimates should be in such a way that the model marginals $P(\boldsymbol{r}_f)$ are equal to the normalised empirical counts:

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

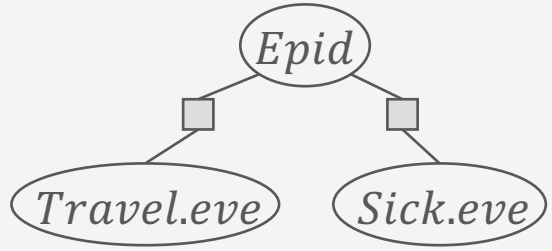
- Does not state how to get the estimates

Factors over Maximal Cliques

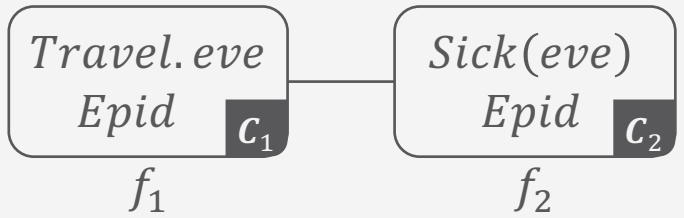
- Markov net with potential functions over its maximal cliques
  - Clique: set of nodes where every node is directly connected with every other node
    - In factor graphs: Directly connected = appear in same factor
  - Maximal clique
    - There is no node in the graph that you could add to the clique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



Markov network (MN)



Factor graph (FG)



Junction tree

Factors over Maximal Cliques

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

$$P_F = \frac{1}{Z}\prod_{f \in F} f = \frac{1}{Z}\prod_{\boldsymbol{C}_i \in V} f_i = \frac{\prod_{\boldsymbol{C}_i \in V} P(\boldsymbol{C}_i)}{\prod_{\{i,j\} \in E} P(\boldsymbol{S}_{ij})}$$

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

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

Factors over Maximal Cliques

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

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

- Set factors to be the normalised empirical counts $P^{\#}(\boldsymbol{r}_f)$ and, for each separator, pick one neighbour and divide the factor by the normalised empirical count of the separator $P^{\#}(\boldsymbol{r}_{ij})$, i.e.,
  - For each $f \in F$,
    - Set $\phi_f(\boldsymbol{r}_f) \leftarrow P^{\#}(\boldsymbol{r}_f)$
  - For each $\{i,j\} \in E$, i.e., separator $\boldsymbol{S}_{ij}$,
    - Choose $h \in \{i,j\}$ at random
    - Set $\phi_h(\boldsymbol{r}_h) \leftarrow \frac{\phi_h(\boldsymbol{r}_h)}{P^{\#}(\boldsymbol{r}_{ij})}$

  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:

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

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

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

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

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

  - Update rule

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

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

Iterative proportional fitting

- Initialise all factors with $t = 0$ uniformly, e.g., all potentials 1
- **for** $t = 1,2, \ldots$ **do**
  - **if** convergence criterion does not hold **then**
    - **for** all $\boldsymbol{r}_f \in \mathcal{R}\big(rv(f)\big), f \in F$ **do**
      - $\phi_f^{(t+1)}(\boldsymbol{r}_f) \leftarrow \phi_f^t(\boldsymbol{r}_f) \frac{P^{\#}(\boldsymbol{r}_f)}{P^t(\boldsymbol{r}_f)}$
    - **else break**

IPF

- Convergence criterion, e.g., given error threshold ε
  - $\forall f \in F : \phi_f^{(t+1)}(\boldsymbol{r}_f) - \phi_f^t(\boldsymbol{r}_f) < \varepsilon$
- Many $P^t(\boldsymbol{r}_f)$ queries to compute over all factors!
  - Efficient execution using a junction tree (jtree)
    - Jtree construction independent of parameters

Radim Jiroušek: Solution of the Marginal Problem and Decomposable Distributions. In: *Kybernetika 27*, 1991.
Radim Jiroušek and Stanislav Přeučil: On the Effective Implementation of the Iterative Proportional Fitting Procedure. In: *Computational Statistics & Data Analysis 19*, 1995.

IPF with Jtrees

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

Compute $P^t(\boldsymbol{r}_f)$ 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 $\boldsymbol{C}_i$ visited enough
- E.g., start at a leaf, traverse the clusters by depth first search

IPF-JT

Yee Whye Teh and Max Welling: On Improving the Efficiency of the Iterative Proportional Fitting Procedure. In: *AISTATS-03 Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

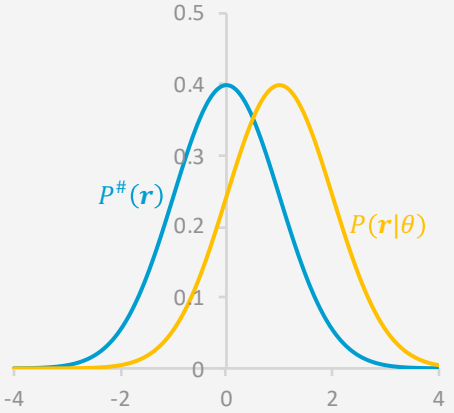
Properties of IPF Updates

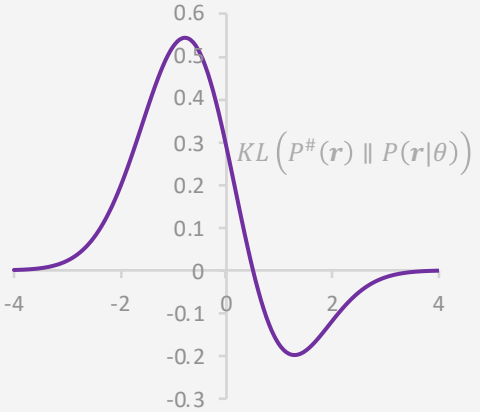
- Reduces to one update per factor if factors are over maximal cliques
- Coordinate ascent algorithm
  - Coordinates = parameters θ in clusters
  - At each step, the update increases the log-likelihood of the data $\log P(D|\theta)$ and it will converge to a global maximum
- Maximising the log-likelihood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

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

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

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





IPF with Incomplete Data

- Problem with incomplete data:
  - Update rule

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

  - Hidden variables do not have $P^{\#}(\boldsymbol{r}_f)$

- EM-IPF: IPF procedure merged with EM scheme
  - E step: Calculate *expected* counts for hidden variables
  - M step: Update parameters $\phi_f^{(t+1)}(\boldsymbol{r}_f)$
  - Could also use jtree for efficiency