Intelligent Agents: Web-mining Agents

Probabilistic Graphical Models

Lifted Learning

Tanya Braun

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Probabilistic Graphical Models (PGMs)

1. Recap: **Propositional** modelling
   - Factor model, Bayesian network, Markov network
   - Semantics, inference tasks + algorithms + complexity
2. **Probabilistic relational models** (PRMs)
   - Parameterised models, Markov logic networks
   - Semantics, inference tasks
3. **Lifted inference**
   - LVE, LJT, FOKC
   - Theoretical analysis
4. **Lifted learning**
   - Recap: propositional learning
   - From ground to lifted models
   - Direct lifted learning
5. **Approximate Inference: Sampling**
   - Importance sampling
   - MCMC methods
6. **Sequential models & inference**
   - Dynamic PRMs
   - Semantics, inference tasks + algorithms + complexity, learning
7. **Decision making**
   - (Dynamic) Decision PRMs
   - Semantics, inference tasks + algorithms, learning
8. **Continuous Space**
   - Gaussian distributions and Bayesian networks
   - Probabilistic soft logic

Outline: 4. Lifted Learning

A. ***Recap: Learning directed models***
   - Maximum likelihood (ML) estimation, expectation-maximisation (EM), structural EM

B. *Learning undirected models*
   - Iterative fitting procedure

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

D. *Statistical relational learning*
   - First-order inductive learning
   - Decision tree representations

Learning BNs

- Known structure, unknown parameters in CPTs
  - Fully observable (complete data)
    - Maximum-likelihood (ML) estimation
  - Hidden (unobservable) variables (incomplete data)
    - Expectation-maximisation (EM)
- Unknown structure
  - Model selection
  - Structural EM

- See also propaedeutic material in Moodle course: Grundlagen des Lernens mit partiellen Daten: Expectation Maximisation (EM) (pdf, pptx, mp4)

General ML procedure

1. Express the likelihood of the data as a function of the parameters to be learned
2. Take the derivative of the log likelihood with respect to each parameter
3. Find the parameter value that makes the derivative equal to 0

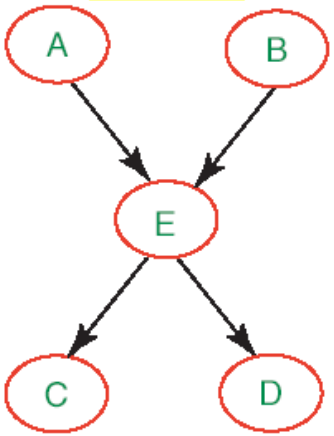
- The last step can be computationally very expensive in real-world learning tasks

Learning BNs: Complete Data

- ML estimation for simplest form of BN learning:
  - Known structure
  - Data containing observations for all variables
    - All variables are observable, no missing data
- Only thing to learn are the network's parameters
  - Entries of the CPTs

  → Count how often which range value occurs given the different values for its parent nodes

Model



Data

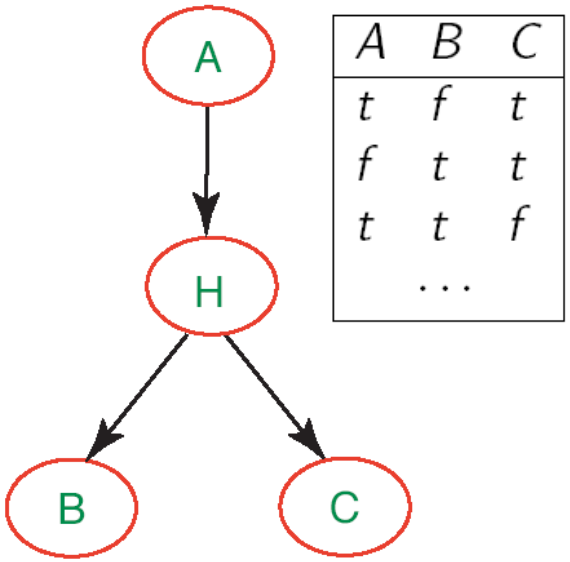
A	B	C	D	E
t	f	t	t	f
f	t	t	t	t
t	t	f	t	f
		...		

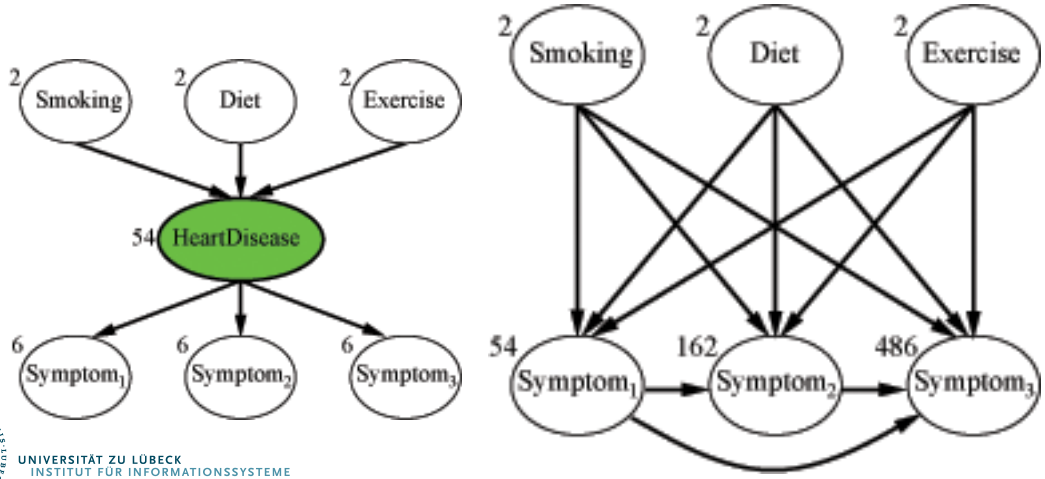
→ Probabilities

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

Learning with Hidden Variables

- If network has hidden variables, no data available for them
  - Cannot count the range values
    - E.g., H occurrences on the left
- Quick fix: Get rid of hidden ones
  - But: may make model much more complex
    - E.g., example below





Range values for all:

$$low, moderate, high$$

Numbers attached to nodes denote how many parameters need to be specified for the CPT of that node

EM: General Idea

- Start from “invented” (e.g., randomly generated) information to solve the learning problem
  - Determine the network parameters
- Refine initial guess by cycling through two steps
  - Expectation (E): update the data with predictions generated via the current model
    - Calculate expected counts
  - Maximisation (M): given the updated data, update the model parameters using MLE
    - Same step compared to learning parameters for fully observable networks
- Full description: see material linked on slide 4

Parameter Estimation

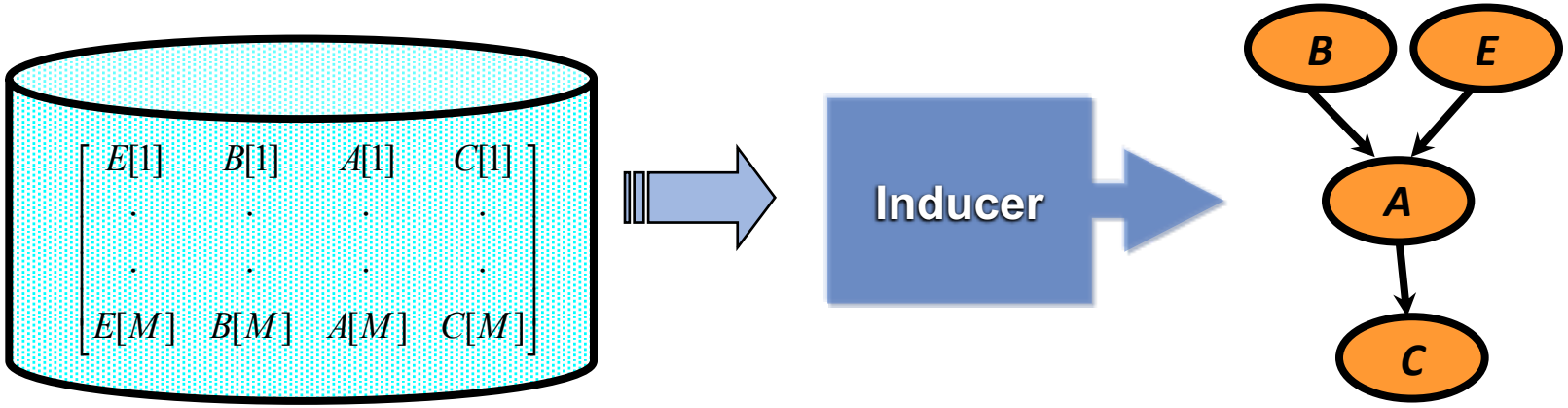
- Assume known structure
- Goal: estimate BN parameters θ given data D
  - θ: CPT entries $P\big(X \mid Parents(X)\big)$
- Parameterisation θ is good if it is likely to generate the observed data $D = \{(x_1, \dots, x_n)_m\}$: ← i.i.d. samples

$$Score(\theta) = P(D|\theta) = \prod_m P\big((x_1, \dots, x_n)_m|\theta\big)$$

- Maximum Likelihood Estimation (MLE) Principle: Choose θ^* so as to maximise $Score$
  - Inherent bias by the assumed structure
    - Only as accurate as the structure allows it to be

Structure Learning

- Unknown network structure
- Given training set D
- Find model that best matches D, includes:
  - *Model selection*
  - Parameter estimation (as on previous slides)



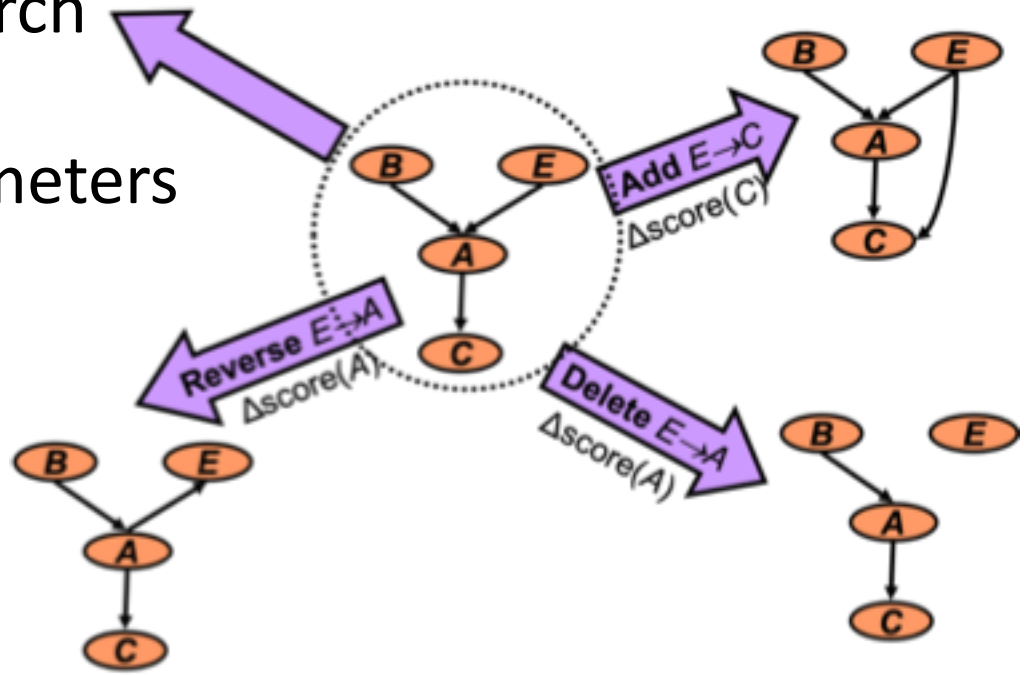
Data D

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

Model selection

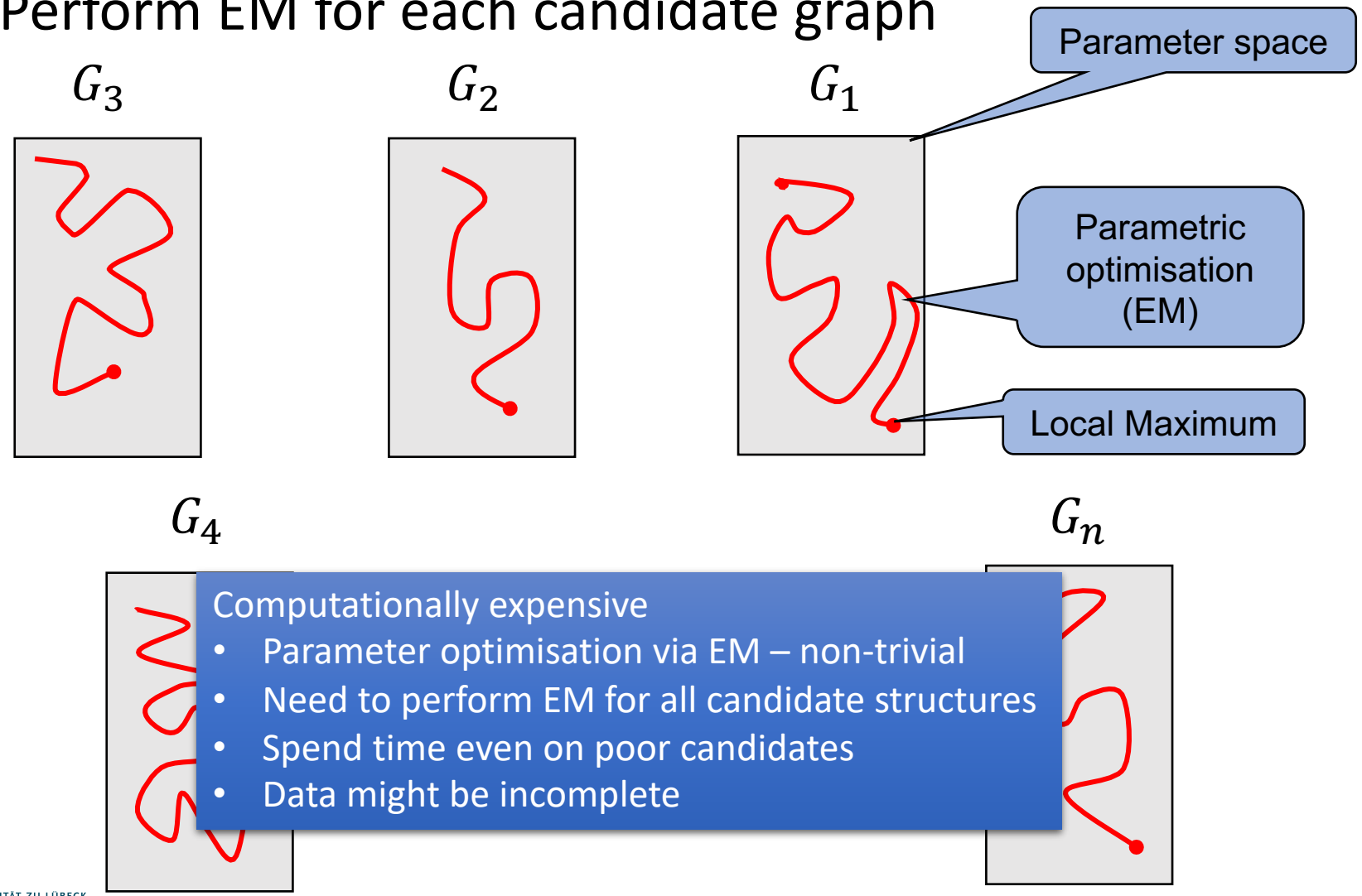
- Goal: Select the best network structure
- Input: Training data, Scoring function
- Output: A network that maximises the score

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

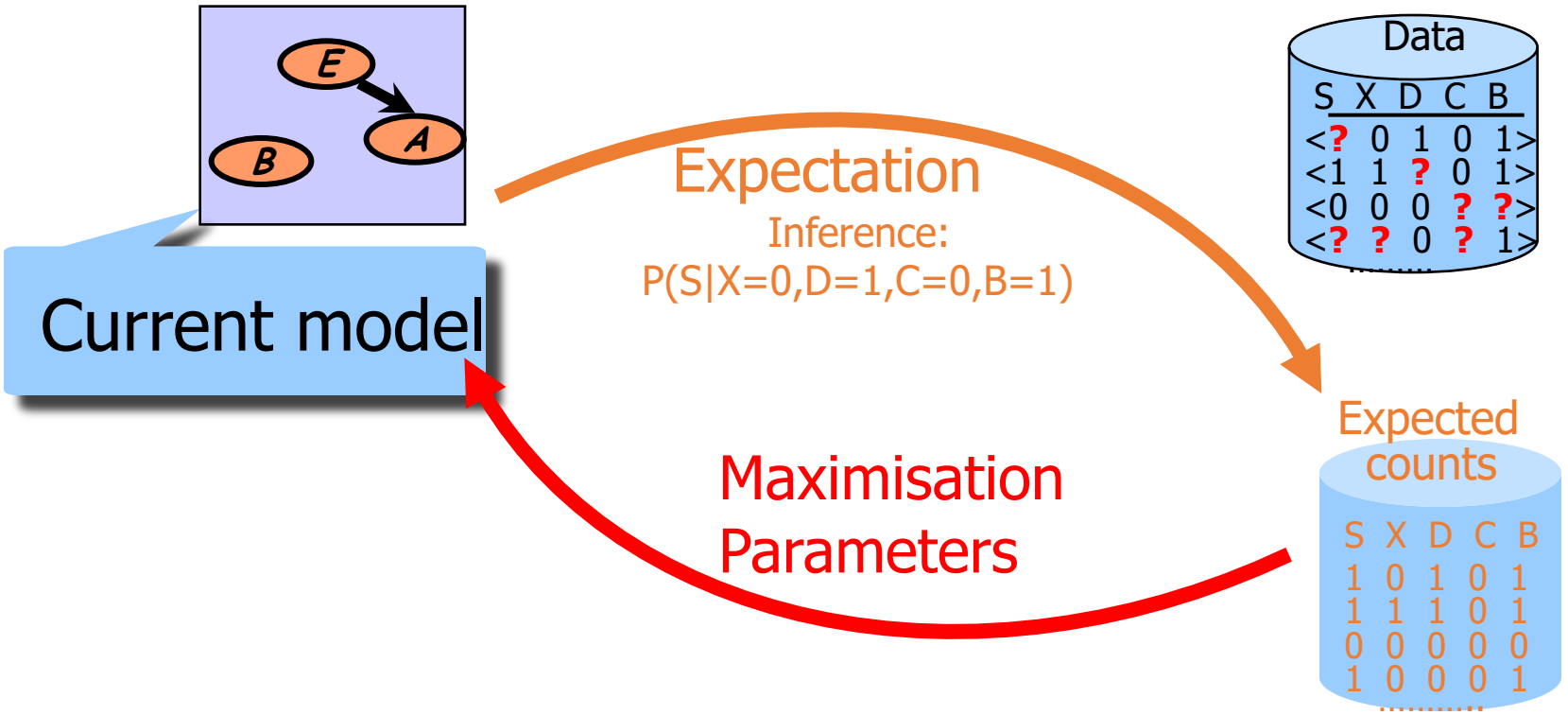
Structure Learning: Incomplete Data

- 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

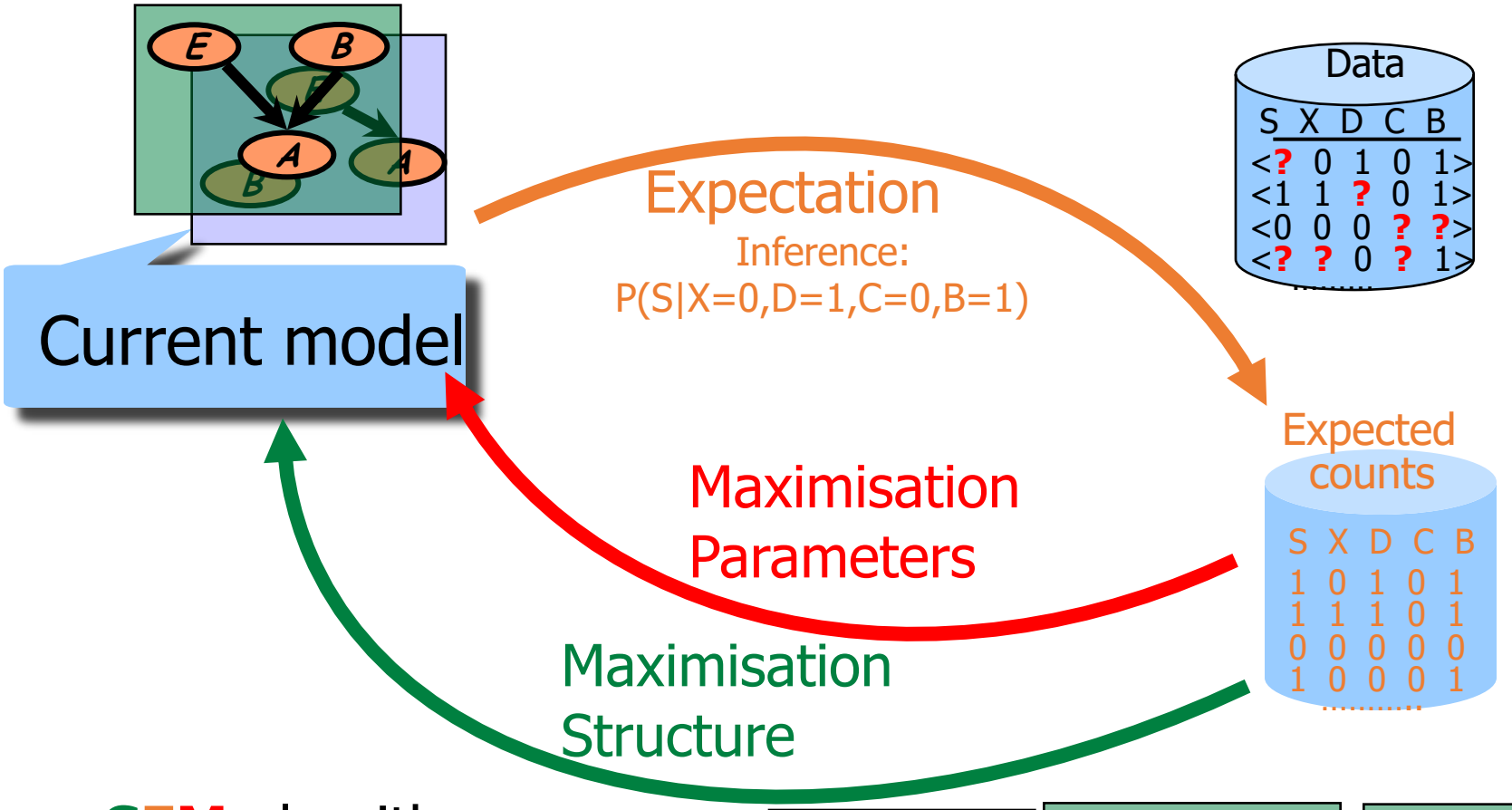
    or
    - Better scoring structure: “structural” EM step

EM

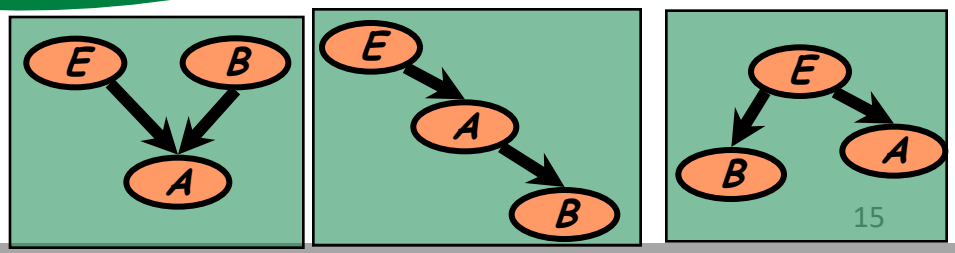


EM-algorithm:
iterate until convergence

Structural EM



SEM-algorithm:
iterate until convergence



Interim Summary

- Known structure, fully observable:
  only need to do parameter estimation
- Known structure, hidden variables:
  use expectation maximisation (EM) to estimate parameters
- Unknown structure, fully observable:
  do heuristic search through structure space, then parameter estimation
- Unknown structure, hidden variables:
  structural EM

Outline: 4. Lifted Learning

A. *Recap: Learning directed models*
   - Maximum likelihood (ML) estimation, expectation-maximisation (EM), structural EM

B. *Learning undirected models*
   - Iterative fitting procedure

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

D. *Relational learning*
   - First-order inductive learning
   - Decision tree representations

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

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

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

  - 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 $r \in \mathcal{R}(R)$
  - Let $W = \times_{i=1}^{l} \mathcal{R}(R_i)$ 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 randvars of some entity α
      - E.g., $\boldsymbol{r}_f = \pi_{rv(f)}(\boldsymbol{r})$: project $\boldsymbol{r}$ onto the randvars 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

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

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

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

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

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

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

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

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

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 $l_2 = k \log(Z)$

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

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

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

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

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

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

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

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

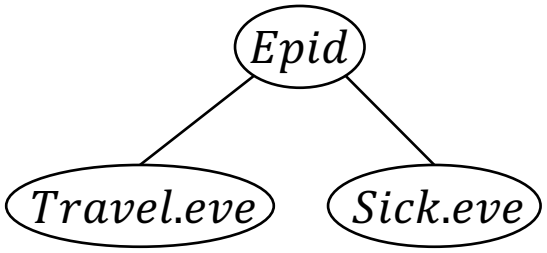
- States that the ML estimates should be in such a way that the model marginals $P(\boldsymbol{r}_f)$ are equal to the normalised empirical counts:

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

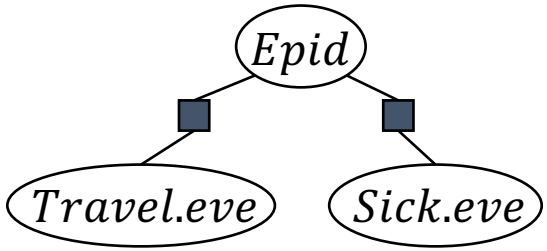
- Does not state how to get the estimates

Factors over Maximal Cliques

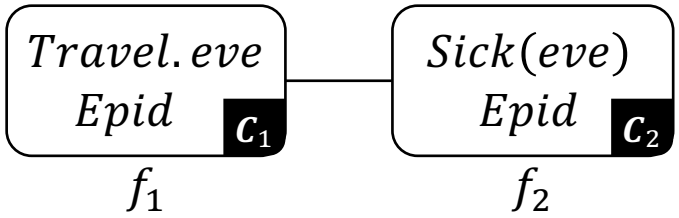
- Markov net with potential functions over its maximal cliques
  - Clique: set of nodes where every node is directly connected with every other node
    - In factor graphs: Directly connected = appear in same factor
  - Maximal clique
    - There is no node in the graph that you could add to the clique 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

$$\forall f \in F : \exists \boldsymbol{C}_i \in V \ : rv(f) = \boldsymbol{C}_i$$
$$\wedge \forall \boldsymbol{C}_j \in V, i \neq j, rv(f) \not\subseteq \boldsymbol{C}_i$$

  → Each $\boldsymbol{C}_i$ has one factor f_i assigned

- 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

- Iterative proportional fitting (IPF) procedure
  - 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 $\boldsymbol{r}_f \in \mathcal{R}(rv(f)), f \in F$ **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)

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

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

- Use jtree to compute $P^t(\boldsymbol{r}_f)$ efficiently
  - Construction independent of parameters
- IPF-JT: IPF with junction trees
  - 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 $\boldsymbol{r}_f \in \mathcal{R}(rv(f)), f \in F_i^t$ **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 $\boldsymbol{C}_j$

IPF-JT

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

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.

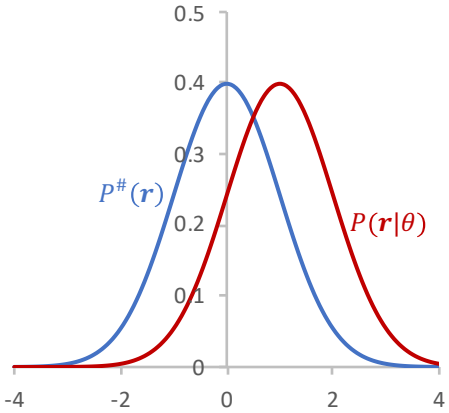
Properties of IPF Updates

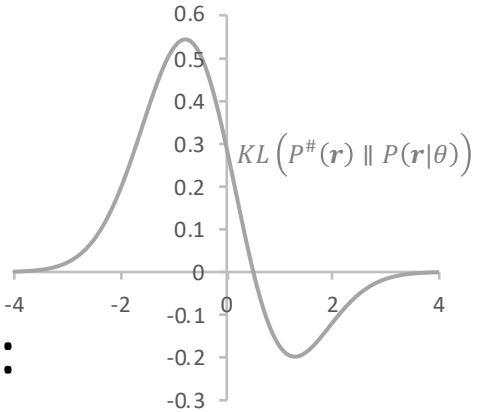
- Reduces to one update per factor if factors are over maximal cliques
- Coordinate ascent algorithm
  - Coordinates = parameters θ in clusters
  - At each step, the update increases the log-likelihood of the data $\log P(D|\theta)$ and it will converge to a global maximum
- Maximising the log-likelihood 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 $P^{\#}(\boldsymbol{r}_f)$

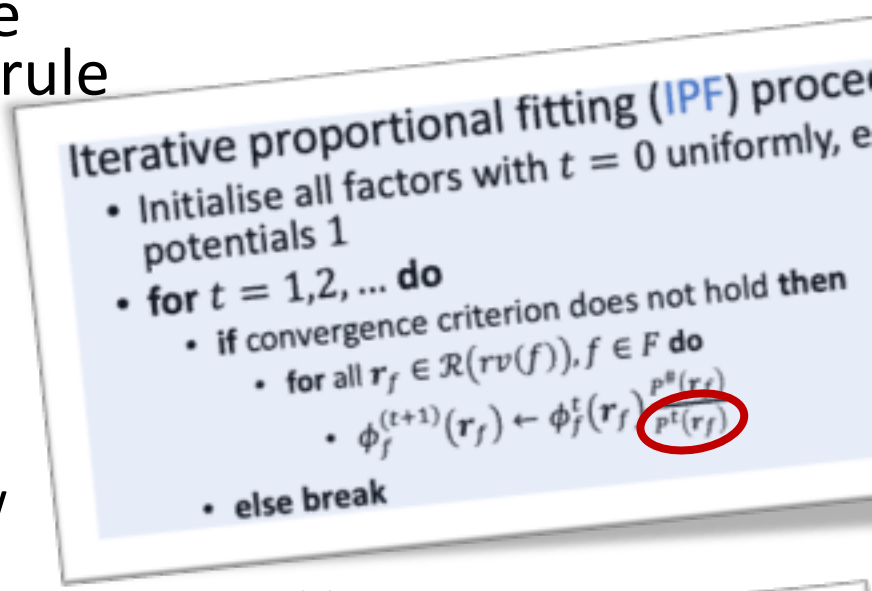
- EM-IPF: IPF procedure merged with EM scheme
  - E step: Calculate *expected* counts for hidden variables
  - M step: Update parameters $\phi_f^{(t+1)}(\boldsymbol{r}_f)$
  - Could also use jtree for efficiency

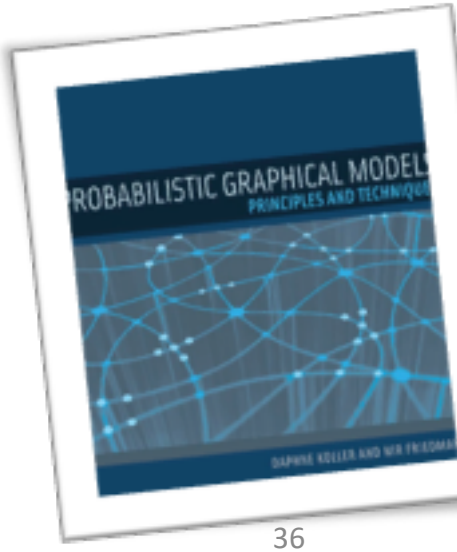
Interim Summary

- Parameter learning in undirected models harder because of $Z \neq 1$
- IPF
  - Fix-point iteration
    - Reduces to one update per factor if factors on maximal cliques
      - Then $Z = 1$ enforced
  - Jtrees for efficiency
    - Update per cluster, then send updated information to neighbour
  - EM version for incomplete data

Addendum to Propositional Learning

- Can use approximate inference to compute queries in update rule
  - E.g., sampling
- Can use alternative objectives
  - Pseudo-likelihood
  - Contrastive divergence
- Structure learning
  - In undirected models can follow the same idea as before
    - Model selection + parameter estimation, possibly interleaved as in structural EM
  - Different approach: independence tests
    - Find which randvars are independent of each other, delete edges accordingly
    - As indicated in first lecture on propositional models





See Chapter 20, in "Probabilistic Graphical Models" by Koller & Friedman (2009) for more details on learning

Generative & Discriminative Models

- Methods described are for learning models representing a full joint probability distribution $P(X, Y)$
  - One can do any kind of inference (prediction, classification, any probability of randvars) one is interested in

→ Generative models

- In contrast: Discriminative models (such as neural nets)
  - Specifically designed and trained to maximise performance of classification: $P(Y|X)$
    - Y a classification randvar 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. *Recap: Learning directed models*
   - Maximum likelihood (ML) estimation, expectation-maximisation (EM), structural EM

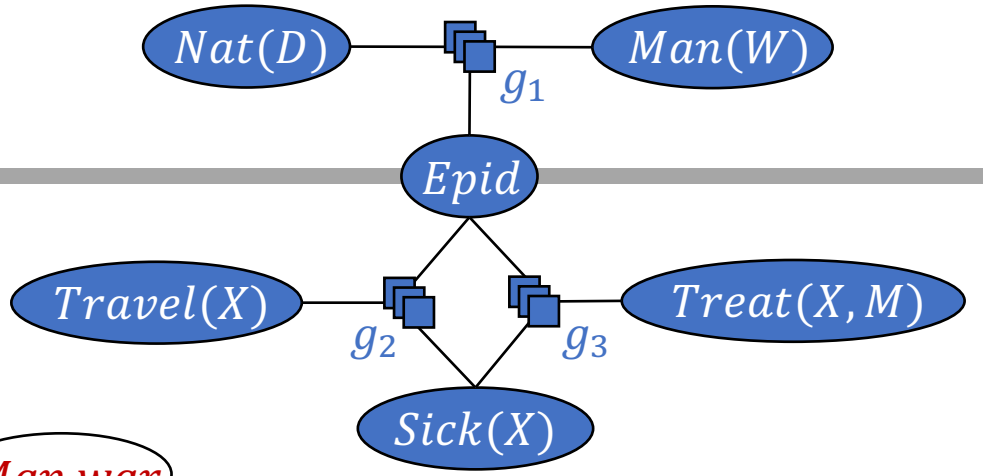
B. *Learning undirected models*
   - Iterative fitting procedure

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

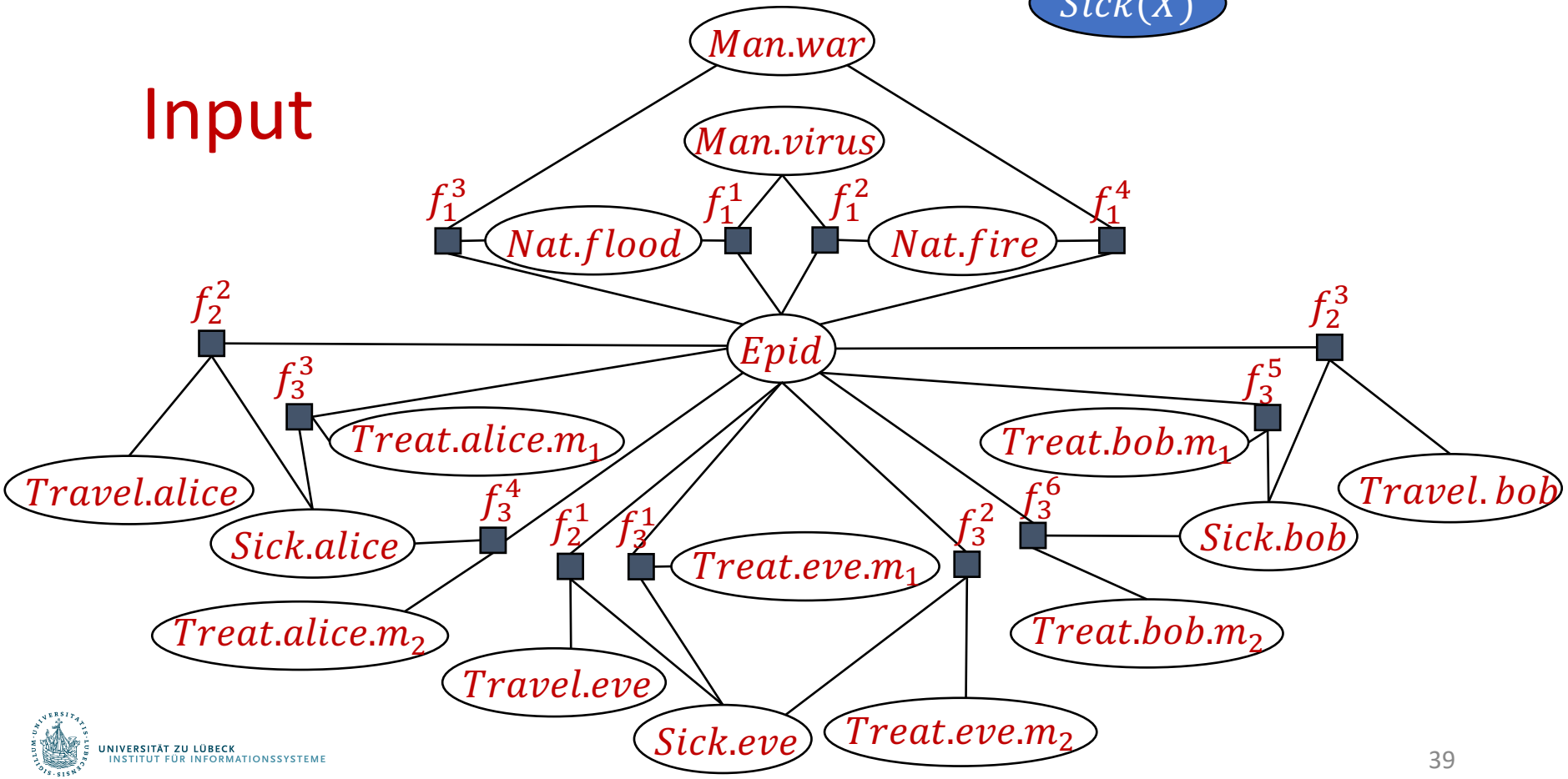
D. *Relational learning*
   - First-order inductive learning
   - Decision tree representations

Compression

Goal

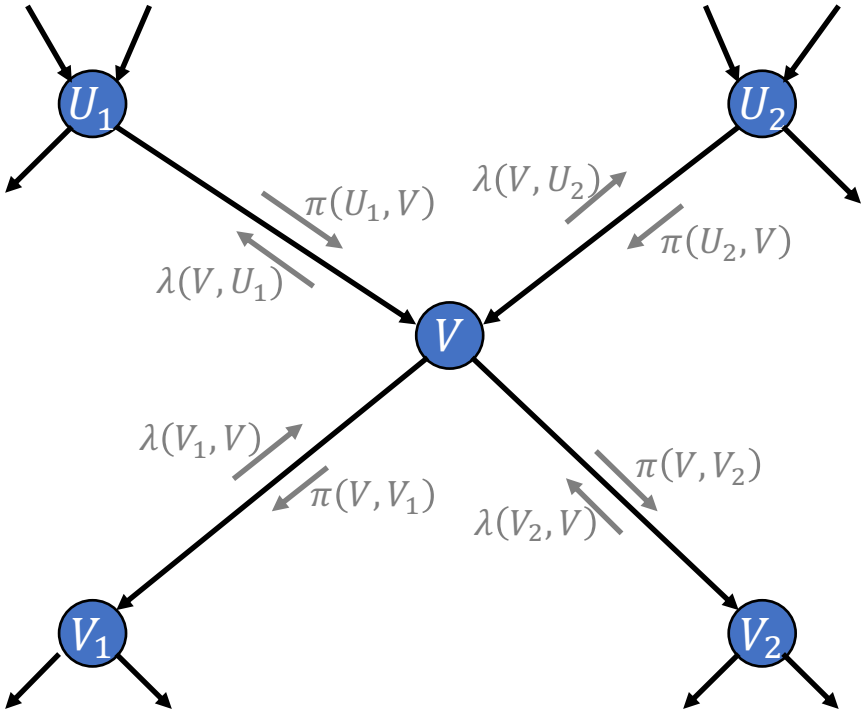


Input



Recap: Foundations of Clustering

- History in propositional probabilistic inference:
  - Based on probability propagation introduced by Pearl (1988)



- If a BN is a polytree, i.e., the underlying undirected graph has no trivial cycles, then
  - Treat each node in a BN as a cluster with the randvars of the accompanying CPT as the cluster randvars
  - Send messages along the edges (to parents and children), eliminating randvars not occurring in the parent or child nodes

Judea Pearl: Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In: *AAAI-82 Proceedings of the 2nd National Conference on Artificial Intelligence*, 1982.

Loopy Belief Propagation

- Pass messages on graph
  - If no cycles: exact
  - Else: approximate

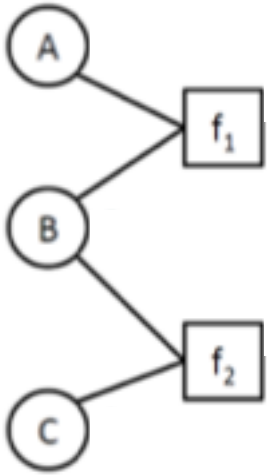
- Lifted (loopy) belief propagation
  - Exploit computational symmetries
  - Compress graph whenever nodes would send identical messages
  - Send messages on compressed graph

→ Colour passing algorithm for compression

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*

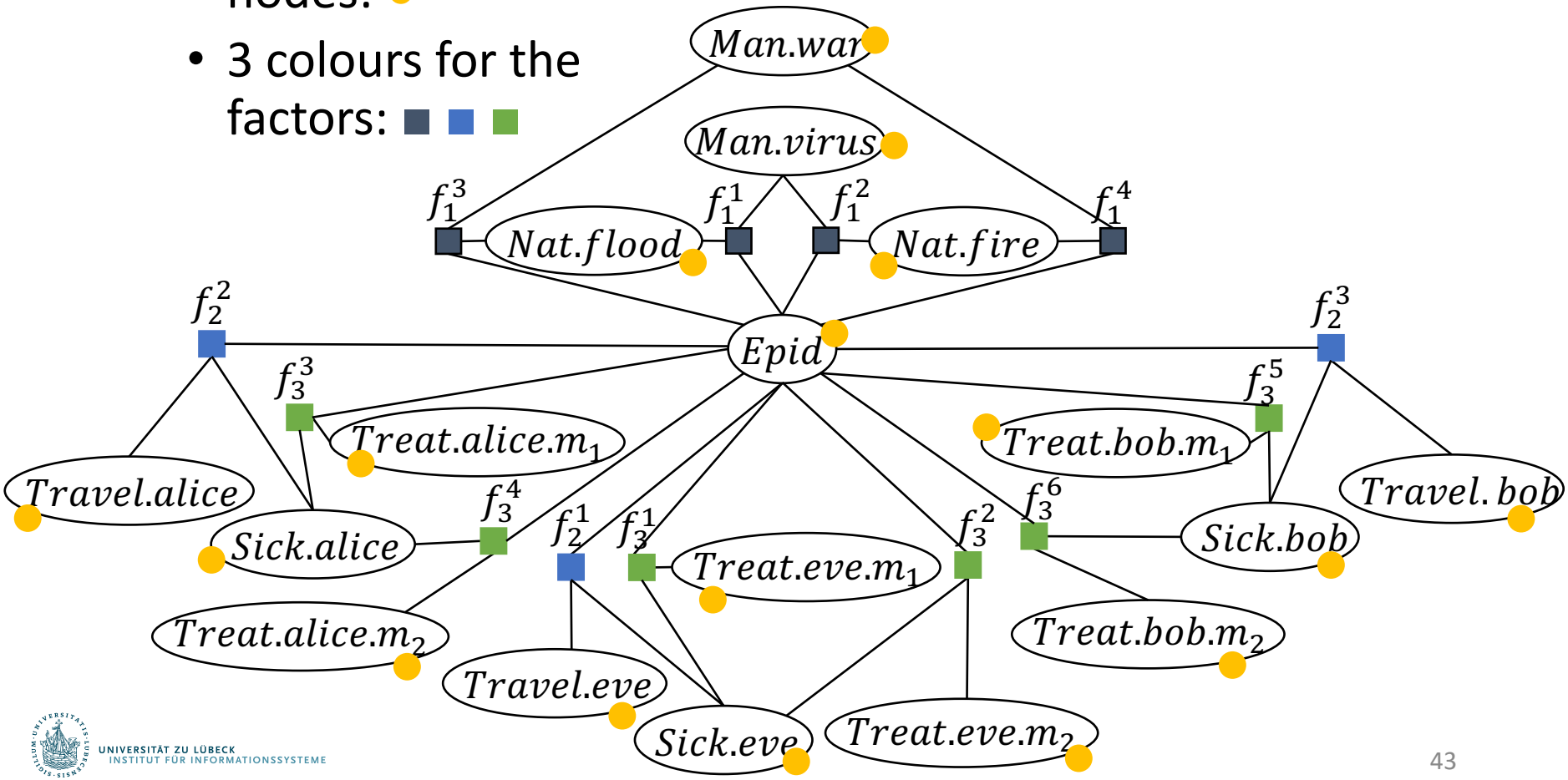
- **Colour nodes according to the evidence you have**
  - No evidence, say **red**
  - State „one“, say **brown**
  - State „two“, say **orange**
  - ...
- **Colour factors distinctively according to their equivalences**
  For instance, assuming f_1 and f_2 to be identical and B appears at the second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



*can also be done at the „lifted“, i.e., relational level

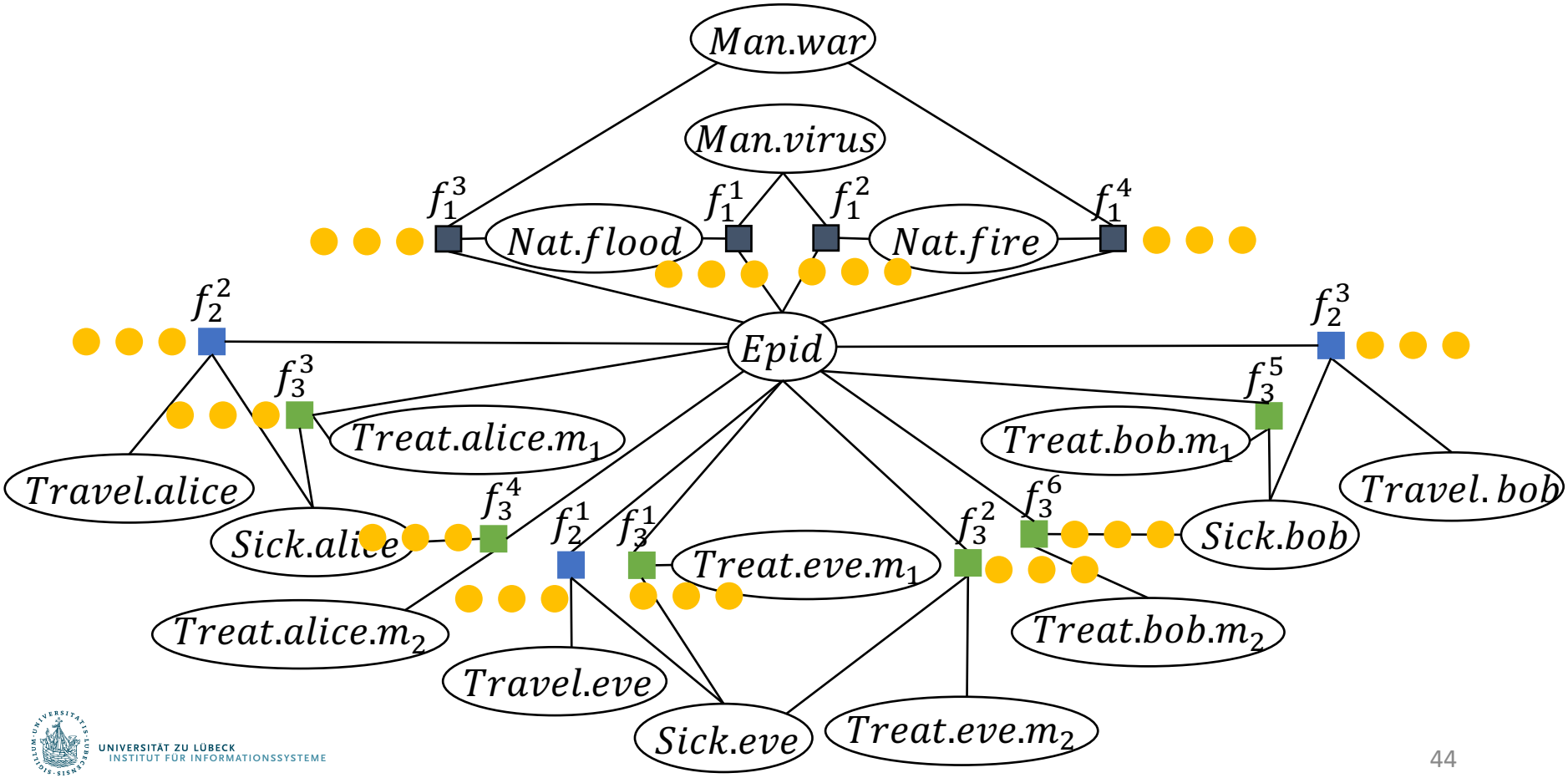
Compression

1. Colour nodes and factors
   - 1 colour for the nodes: ●
   - 3 colours for the factors: ■ ■ ■



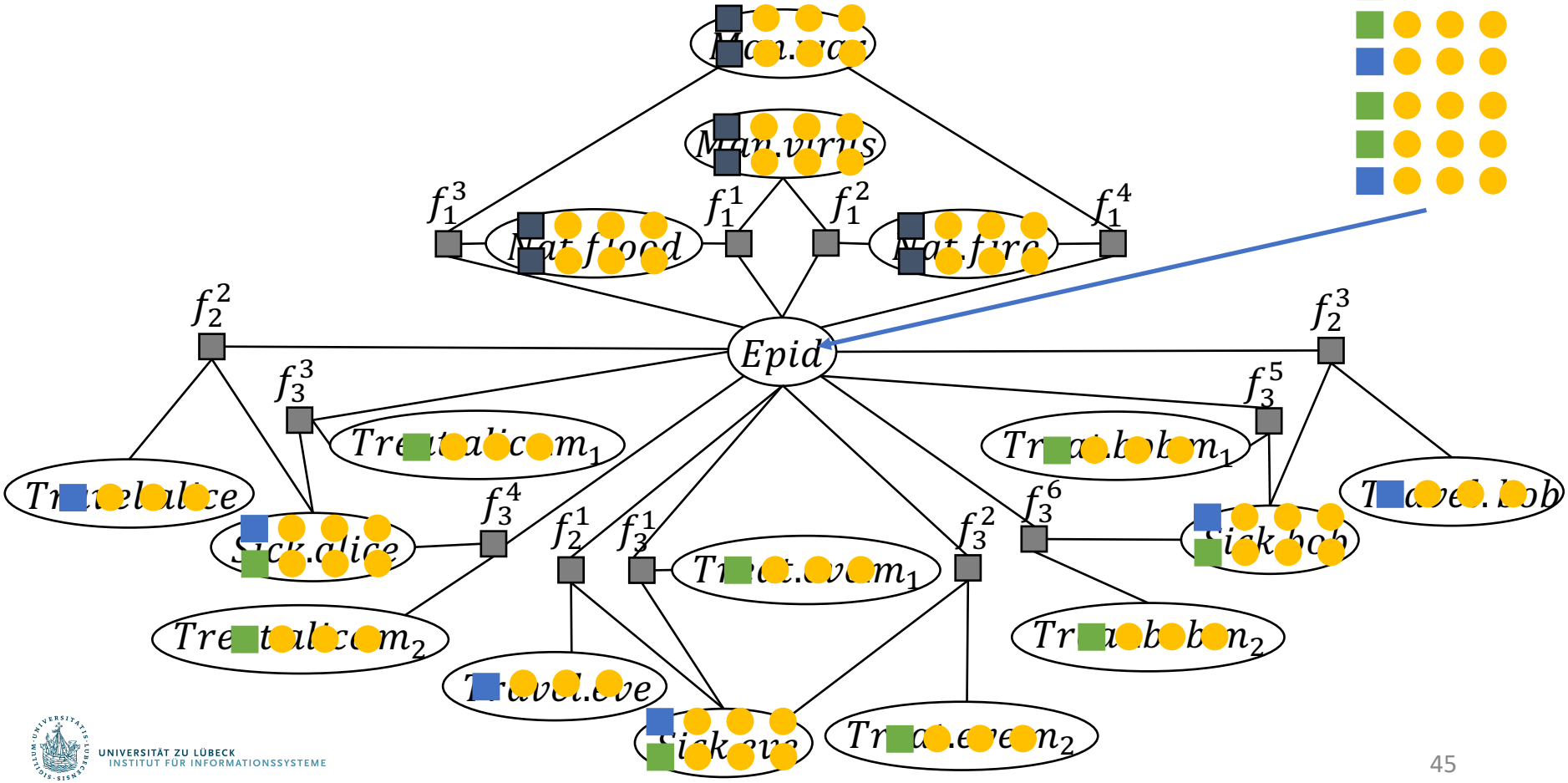
Compression

2. Factors collecting colours from nodes, signing their own colours to the collected ones



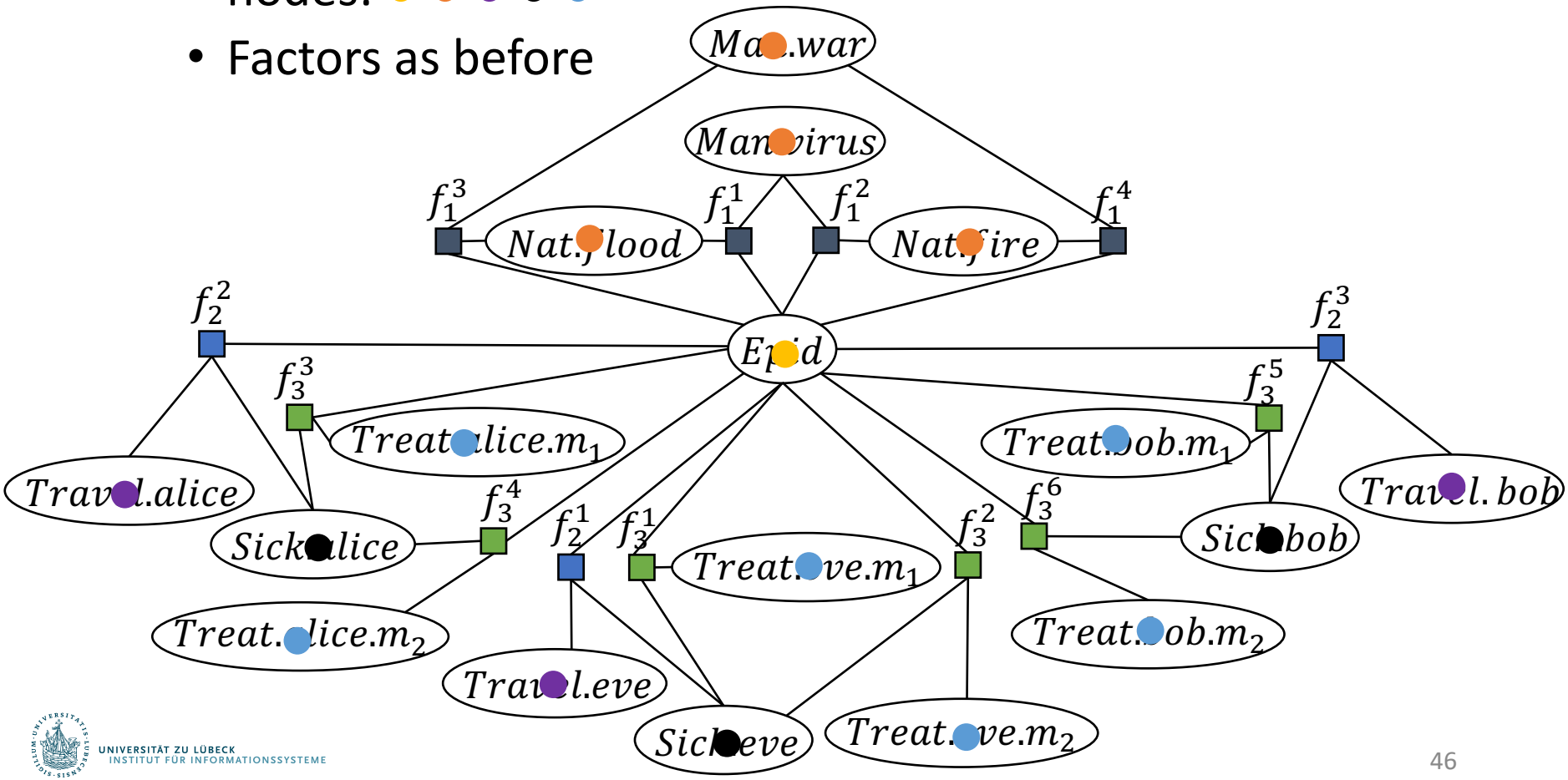
Compression

3. Nodes collecting colours from factors



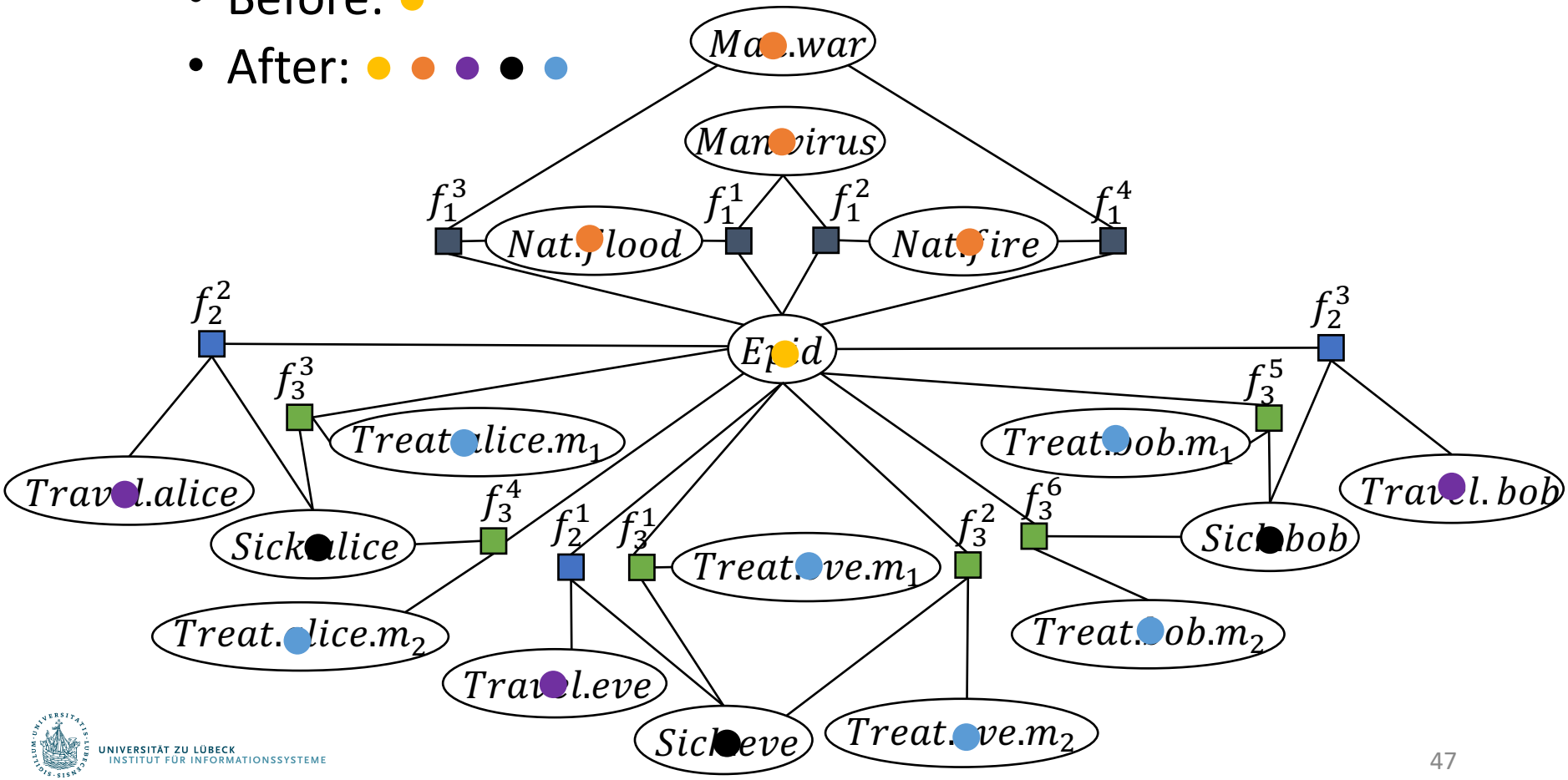
Compression

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



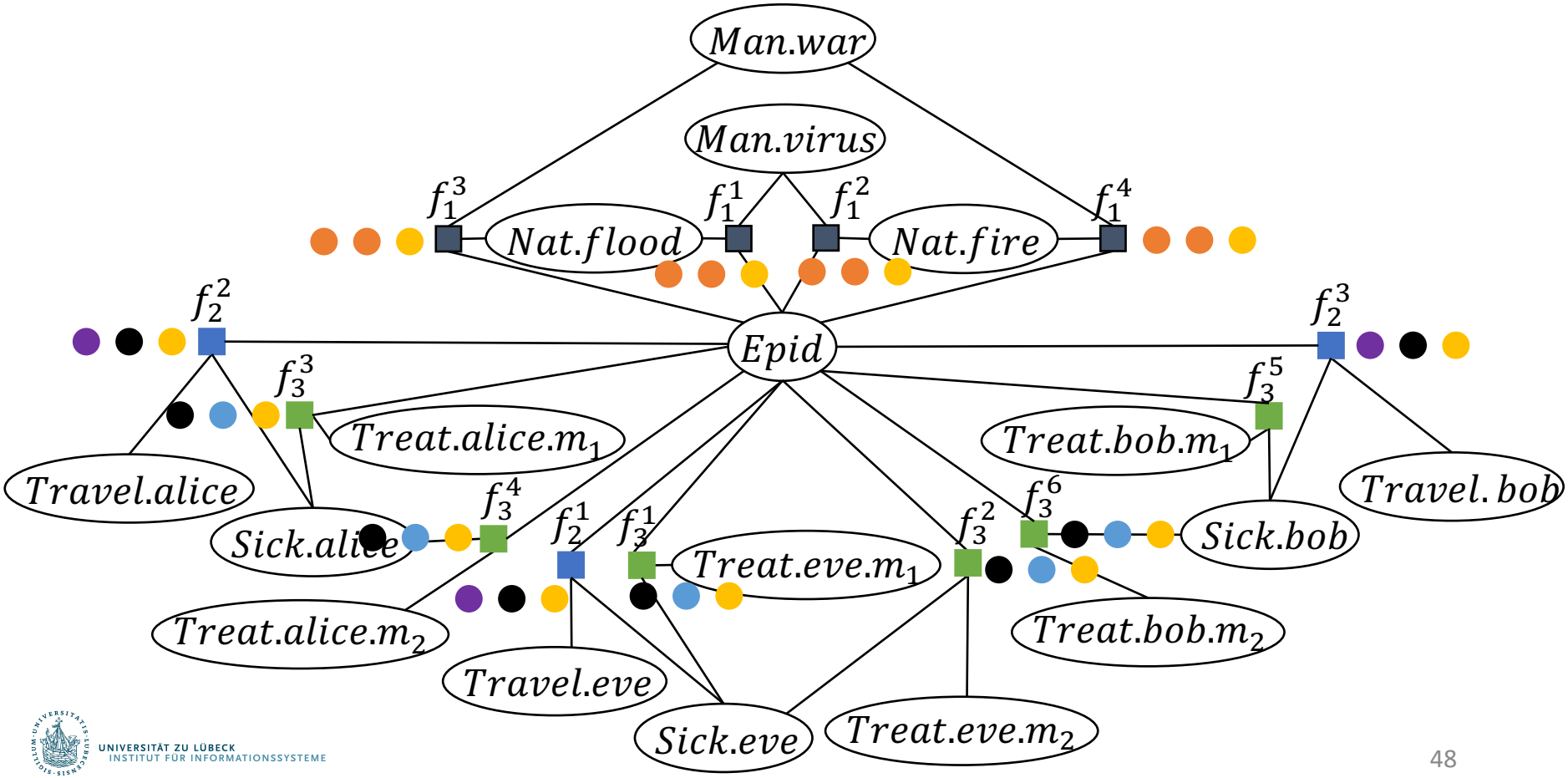
Compression

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



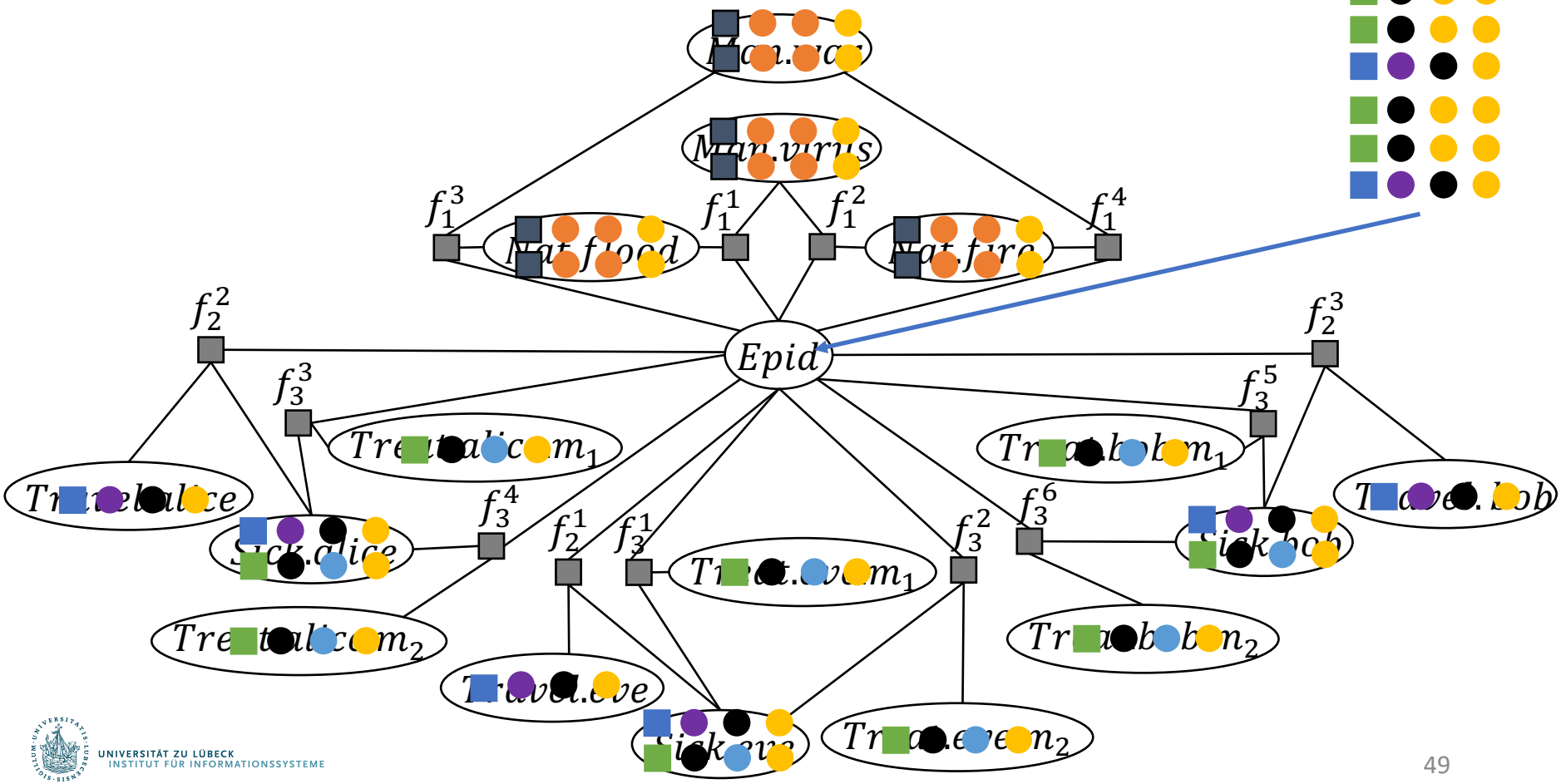
Compression

2. Factors collecting colours from nodes, signing their own colours to the collected ones



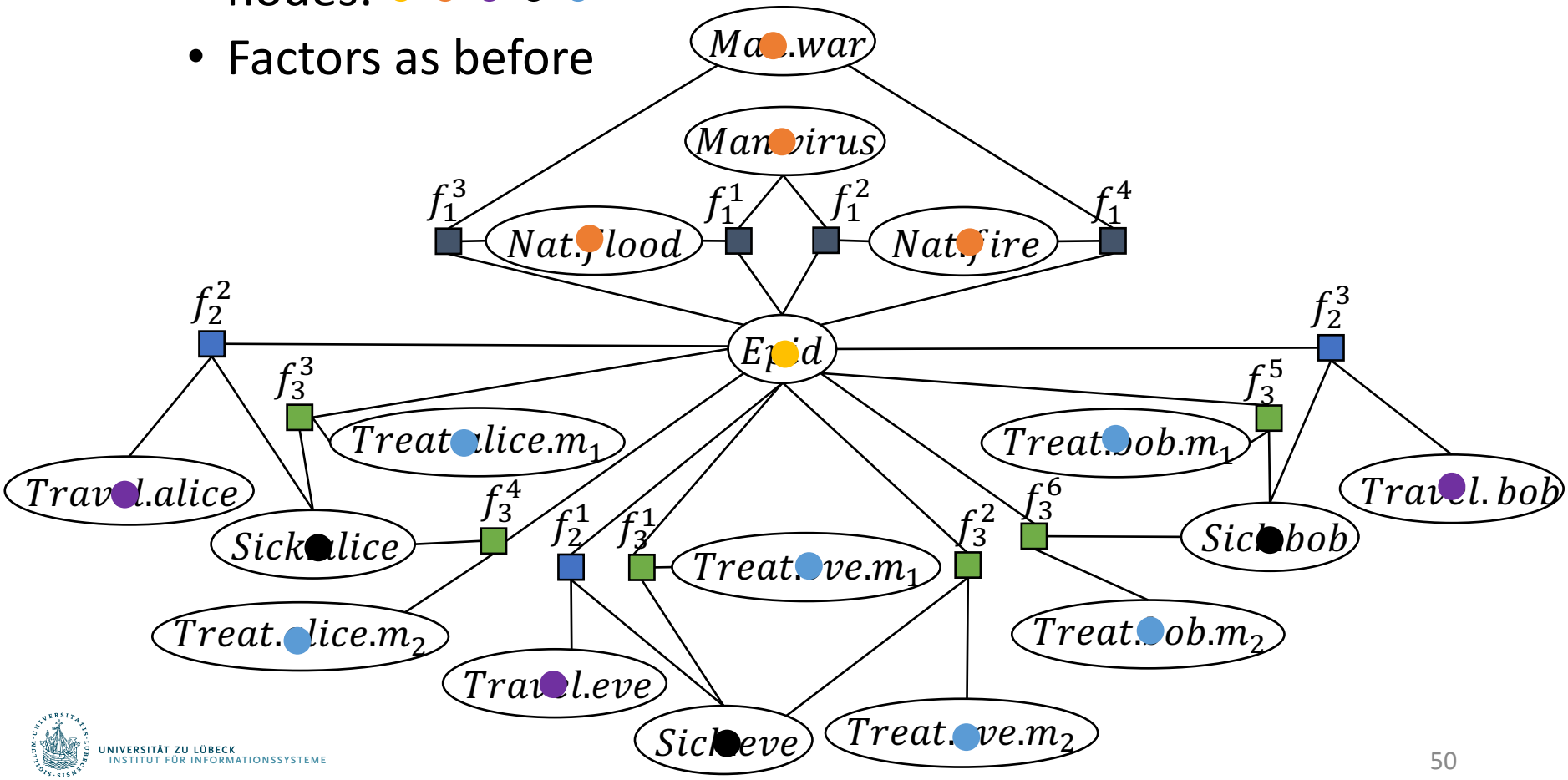
Compression

3. Nodes collecting colours from factors



Compression

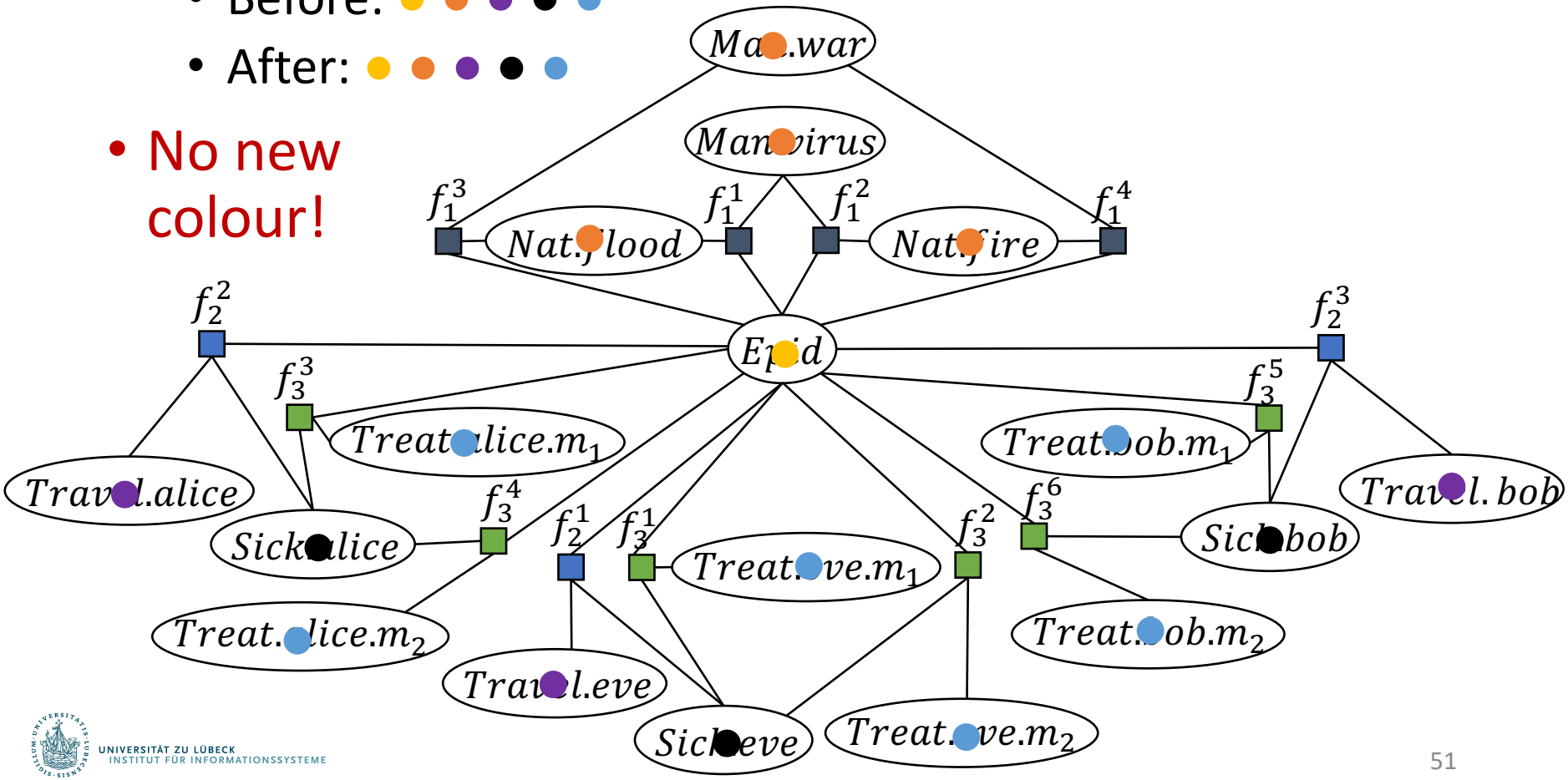
4. Recolour nodes based on collected signatures
   - 5 colours for the nodes: ● ● ● ● ●
   - Factors as before



Compression

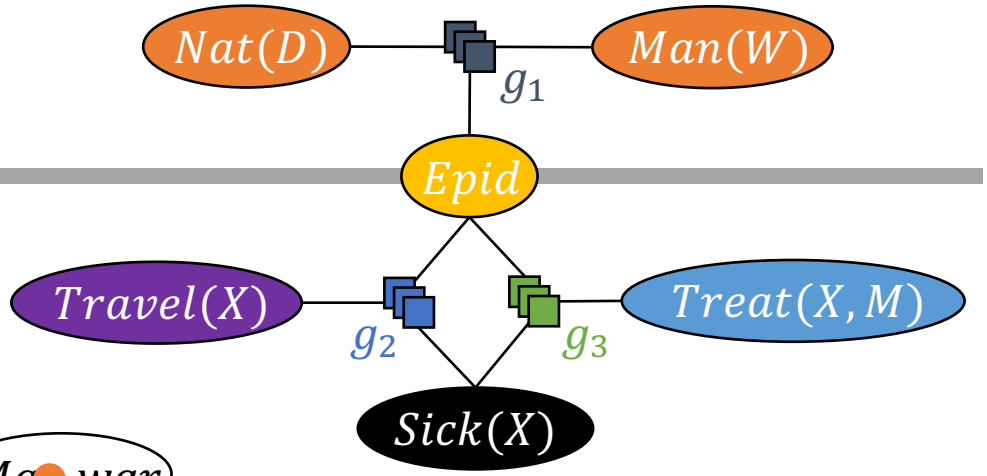
5. If no new colour created, stop. Otherwise, pass colours again.
   - Before: ● ● ● ● ●
   - After: ● ● ● ● ●

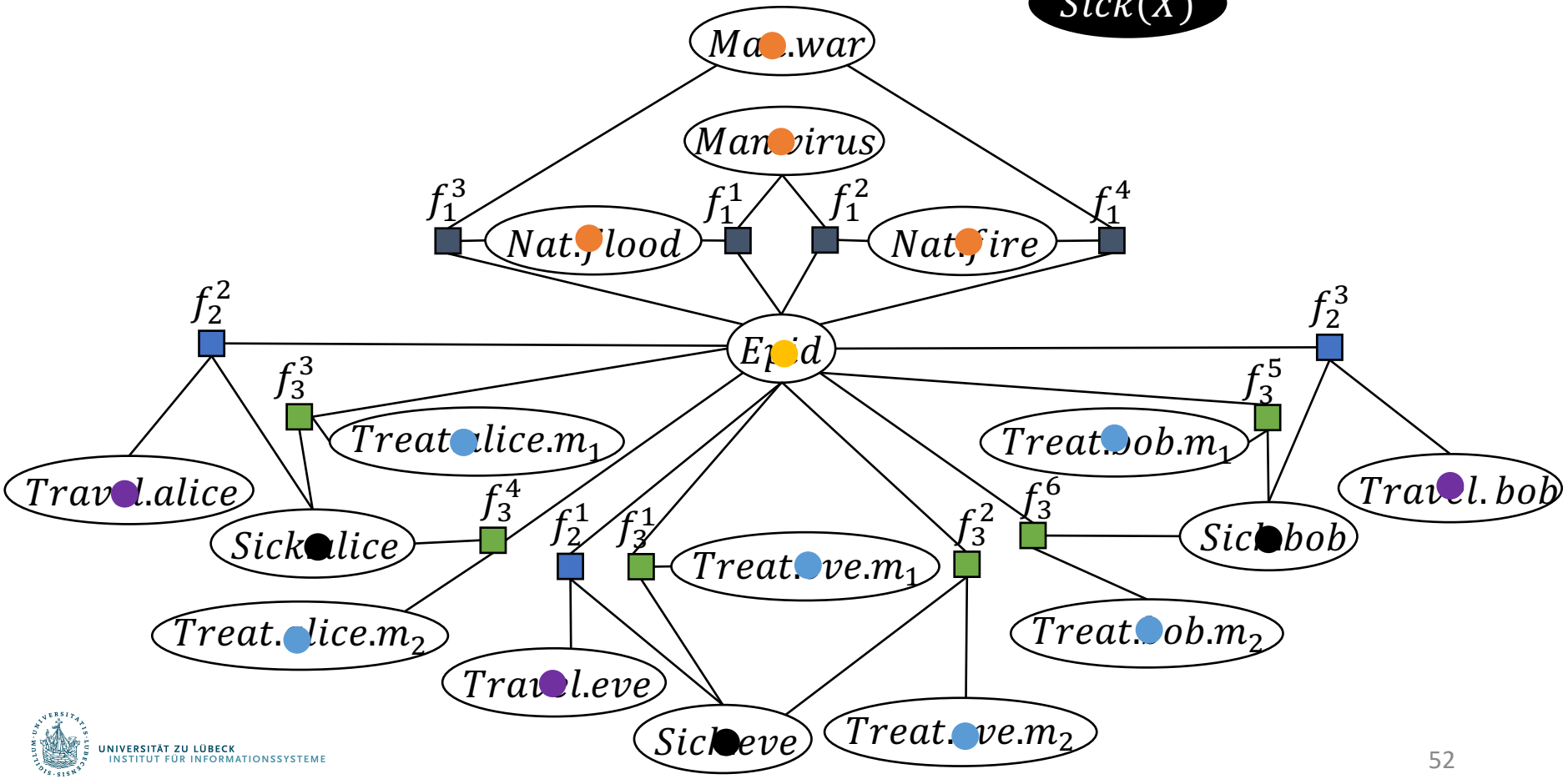
- No new colour!



Compression

- Compressed graph:





Colour Passing Compression

- Algorithm:
  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 randvars of same colour using logvars
- 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 randvars
- "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. *Recap: Learning directed models*
   - Maximum likelihood (ML) estimation, expectation-maximisation (EM), structural EM

B. *Learning undirected models*
   - Iterative fitting procedure

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

D. *Relational learning*
   - First-order inductive learning
   - Decision tree representations

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 randvars but also logvars 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

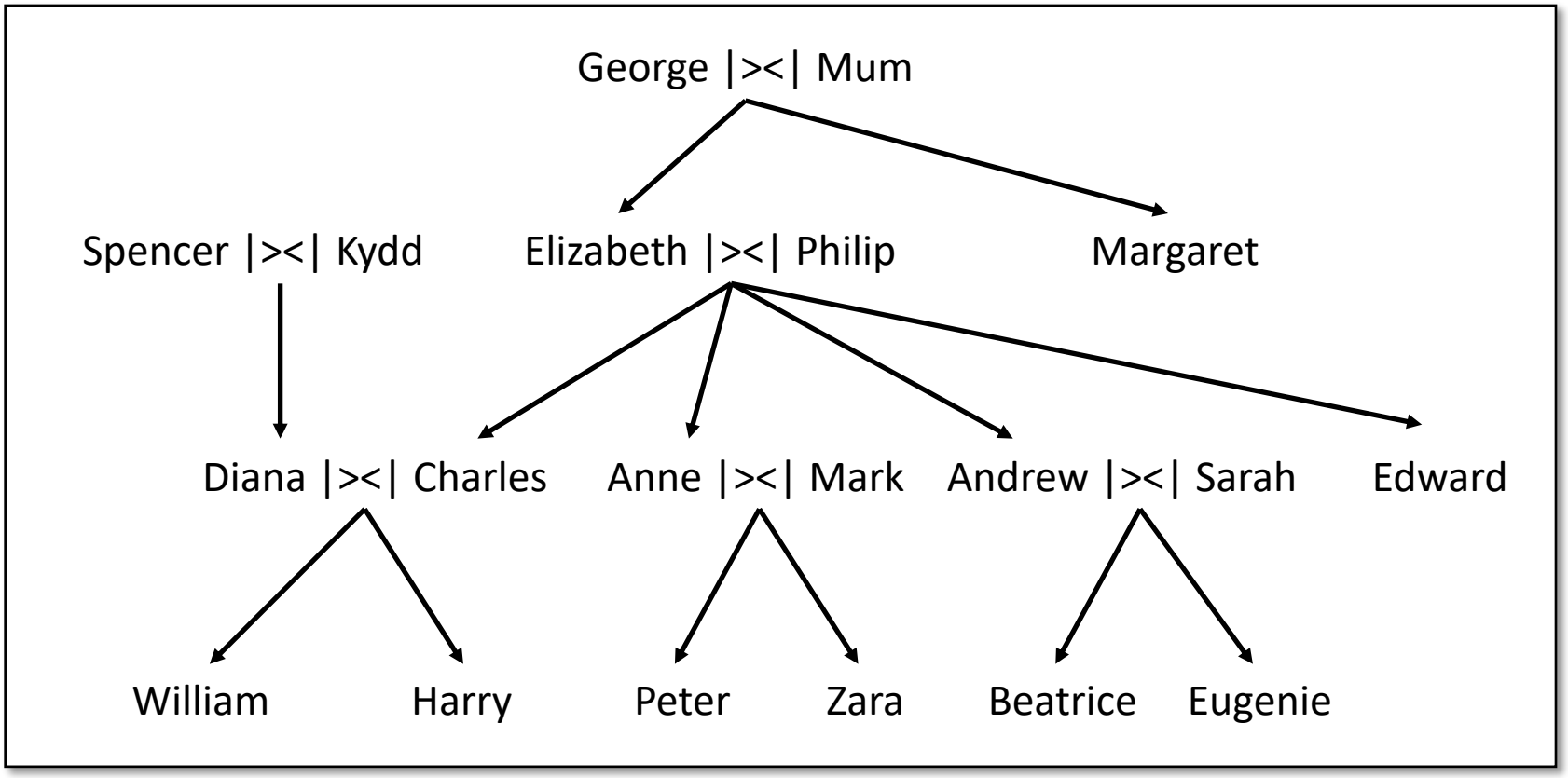
- Logic perspective on learning
  - Examples are composed of descriptions and classifications
  - Objective is to find a hypothesis that explains the classification of the examples, given their descriptions
    - $\text{Hypothesis} \wedge \text{Descriptions} \models \text{Classifications}$
- Knowledge-based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

First-order Inductive Learning (FOIL)

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

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

A Not Up-to-date Example



Example Continued: *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

- A little bit of background knowledge helps a lot

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

  - 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 the positive examples as instances of the $Grandfather$ relationship
  - Start with a clause with an empty body
  - All examples are now classified as positive, so specialize to rule out the negative examples
  1. Incorrectly classifies the 12 positive examples
  2. Incorrect on a larger part of the negative examples
  3. Prefer the third clause and specialise it further

- 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 a clause that covers 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

- 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: $Mother(z, u)$, $Married(z, z)$, $Grandfather(v, x)$
    - Invalid: $Married(u, v)$
  - Inequality: $z \neq x$
  - Arithmetic comparisons: $x > y$ (not meaningful)
- Approach: Add to $literals$
  - Using predicates:
    - Negated or unnegated
    - Use any existing predicate (including the goal)
      - By allowing the target predicate at this point, FOIL is able to learn recursive definitions, but has to be kept 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 equality and for inequality 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
      - cf. 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

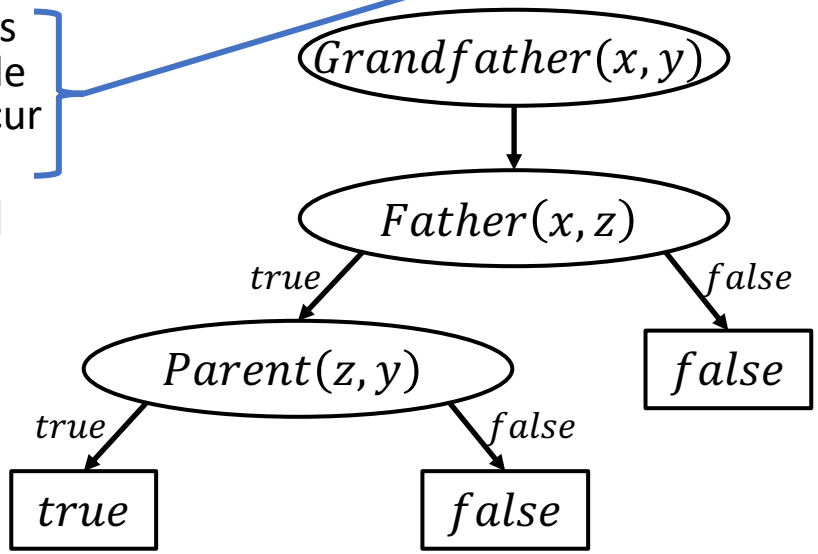
FOIL: Optimisations

- The way $\mathrm{New-Literal}$ 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 the clause becomes longer than the total length of the positive examples that the clause explains, this 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



More on decision trees: Topic 7 here

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.

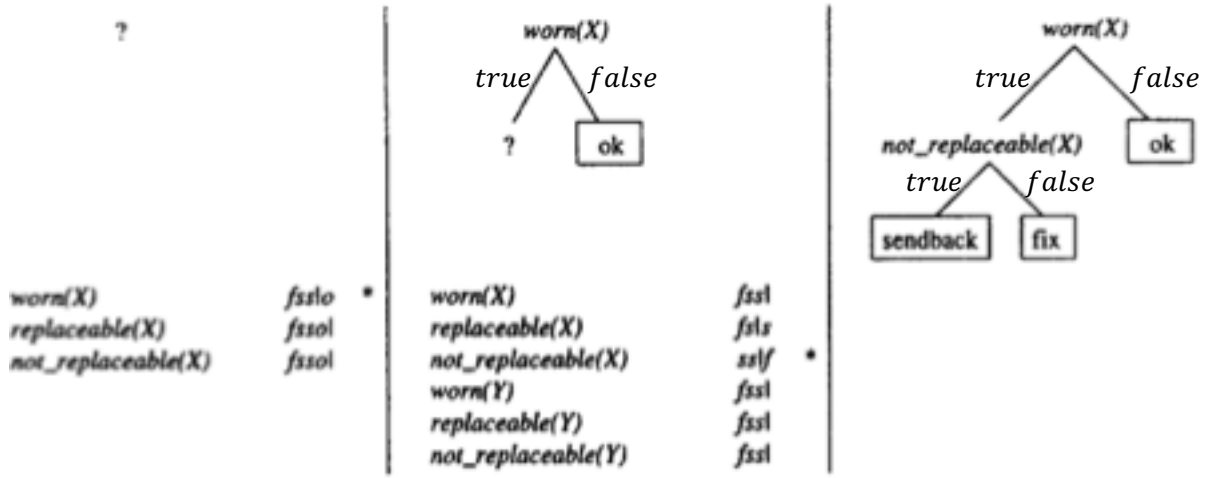
FOLDT: Example

- 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

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)



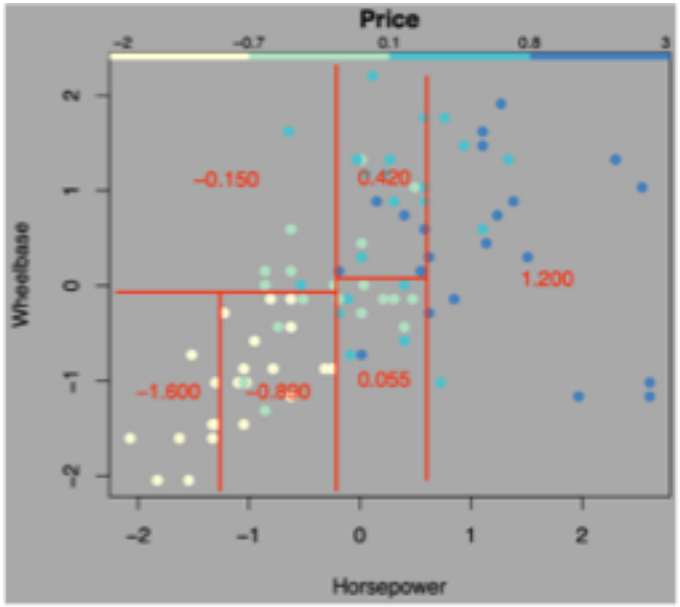
- Resulting logic program:

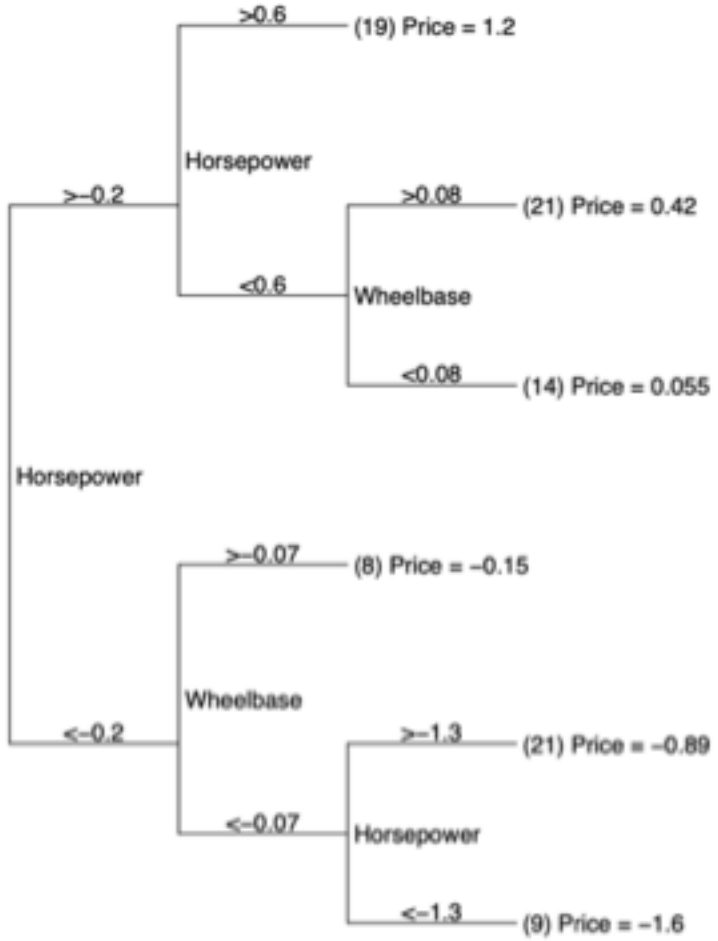
```
class(ok)        ← ∀X : ¬worn(X)
class(sendback)  ← ∃X : worn(X) ∧ not_replaceable(X)
class(fix)       ← ∃X : worn(X) ∧
                   ∀Y : (¬worn(Y) ∨ ¬not_replaceable(X))
```

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



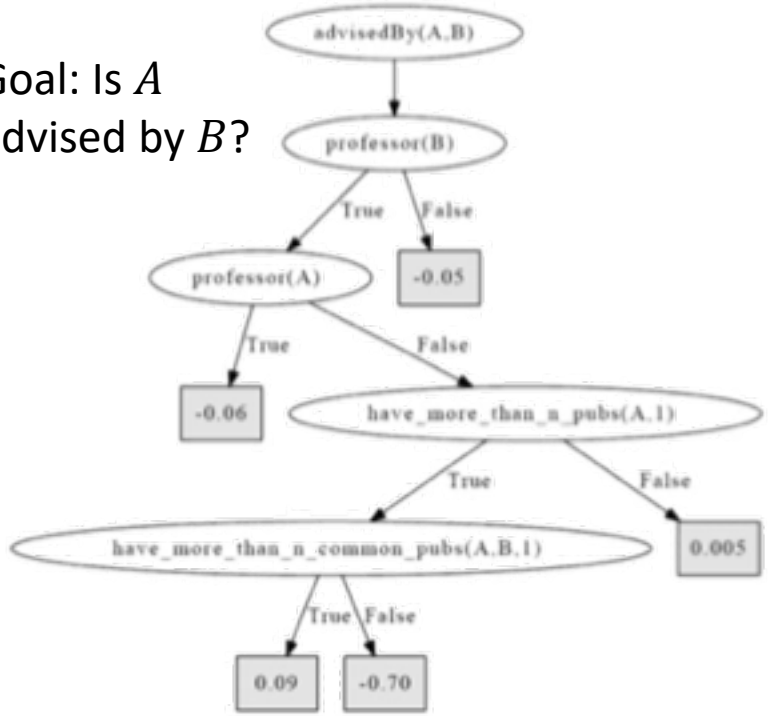


http://www.stat.cmu.edu/~cshalizi/350-2006/lecture-10.pdf

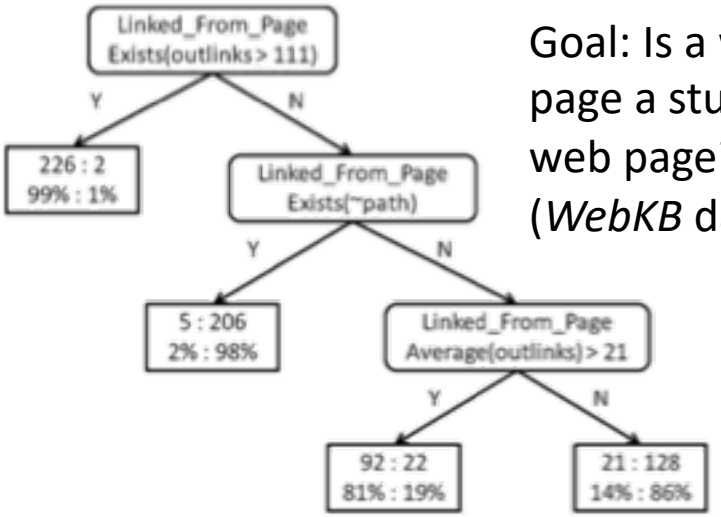
Relational Regression & Probability Trees

- 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/

Learn Approximate Models

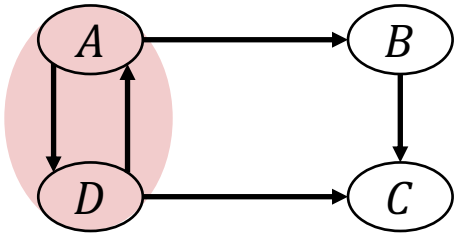
- Relational dependency networks (RDNs)
  - Using RPTs for local distributions in RDNs
  - Boosted learning: Learn a set of distributions to approximate a local distribution
    - Set of RRTs for local distributions in RDNs
  - Based on approximate propositional model of dependency networks (DN)
- Next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

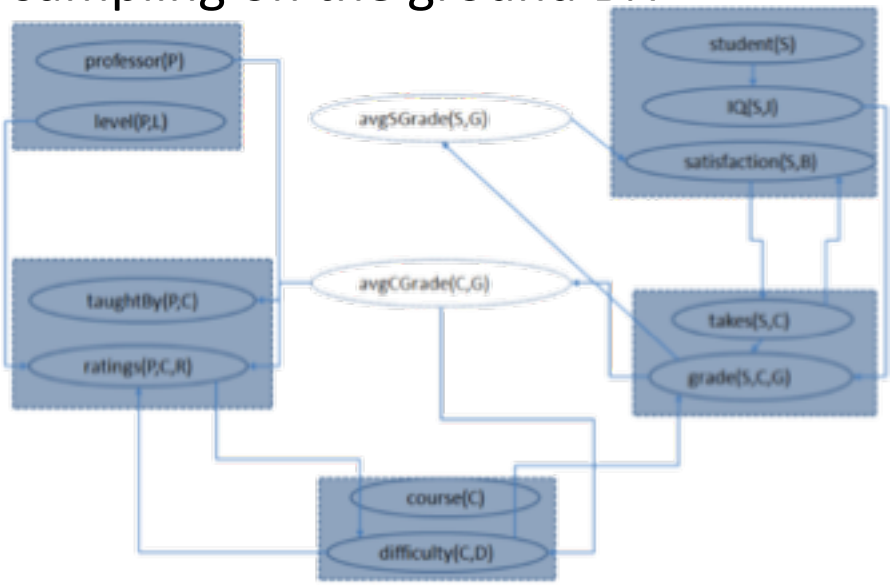
$P(A|D)$
$P(B|A)$
$P(C|B,D)$
$P(D|A)$

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
- Each predicate A_k associated with a CPD $P\left(A_k | parents(A_k)\right)$
  - Aggregators such as count, max, average

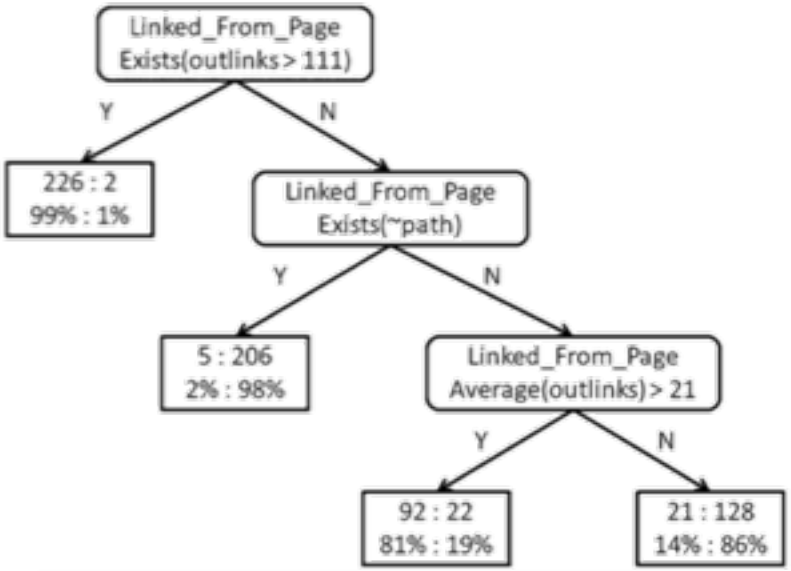
$$P_F(\boldsymbol{r}_f) = \frac{\prod_{f \in F} P(\boldsymbol{r}_f)}{\prod_{\{i,j\} \in E} P(\boldsymbol{r}_{ij})}$$



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

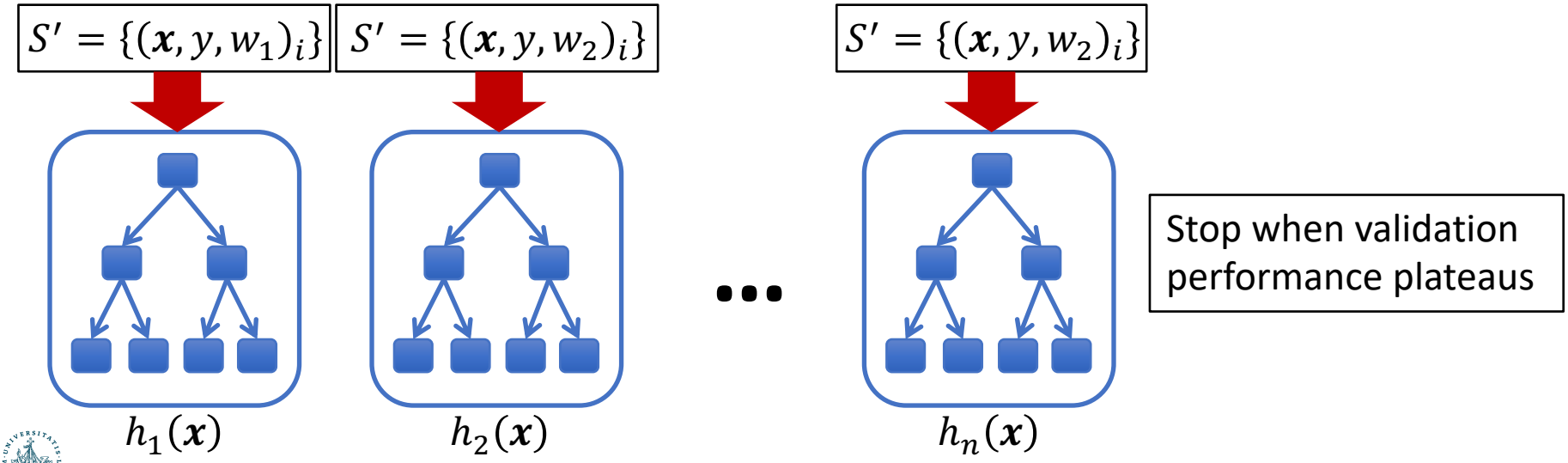
- Used in hypothesis testing to test, e.g., if observed values follow a theoretical distribution; given α, often $\alpha = 0.05$, if $p < \alpha$, reject H_0

(More: Topic 5 here)

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

Boosting Idea

- Use an *ensemble* of classifiers ($\boldsymbol{x}$: feature vector, y: class labels)
  - Each classifier marginally better than random guessing
    - Idea: each (simple) classifier works well enough for a subset of the samples but not all of them; loop:
      - Train a classifier $h_i(\boldsymbol{x})$ on the 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'
  - Combine these weak classifiers to one strong classifier $h(\boldsymbol{x})$
    - Using a weighted sum: $h(\boldsymbol{x}) = \sum_i \alpha_i h_i(\boldsymbol{x})$
  - E.g., *AdaBoost* with decision trees: $h(\boldsymbol{x}) = \alpha_1 h_1(\boldsymbol{x}) + \alpha_2 h_2(\boldsymbol{x}) + \cdots + \alpha_n h_n(\boldsymbol{x})$



(more on ensemble methods: Topic 8 here)

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 it.)

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

- Set of $\Delta_m(y_i; \boldsymbol{x}_i)$ over all i form set of training examples

→ Train a function h_m that approximates $\Delta_m(y_i; \boldsymbol{x}_i)$

→ 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 $P\big(Y|parents(Y)\big)$ for all predicates Y
- Let $P\big(y|parents(y)\big)$ for a grounding y of Y be

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

  where
  - $\forall x \in \boldsymbol{x}, x \neq y, \psi(y; \boldsymbol{x})$ denotes potential function of y given all other $x, x \neq y$ and y' iterates through all possible 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

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

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

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

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

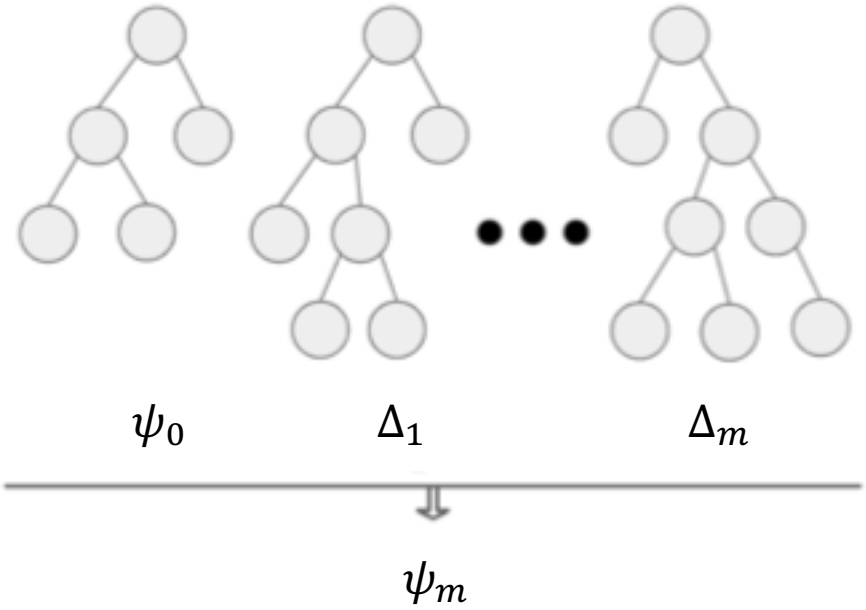
  - Gradient at each example: adjustment required for the probabilities to match the observed value for that example
  - Use $y_i; \boldsymbol{x}_i$ and $\Delta_m(y_i; \boldsymbol{x}_i)$ for all examples to fit an RRT

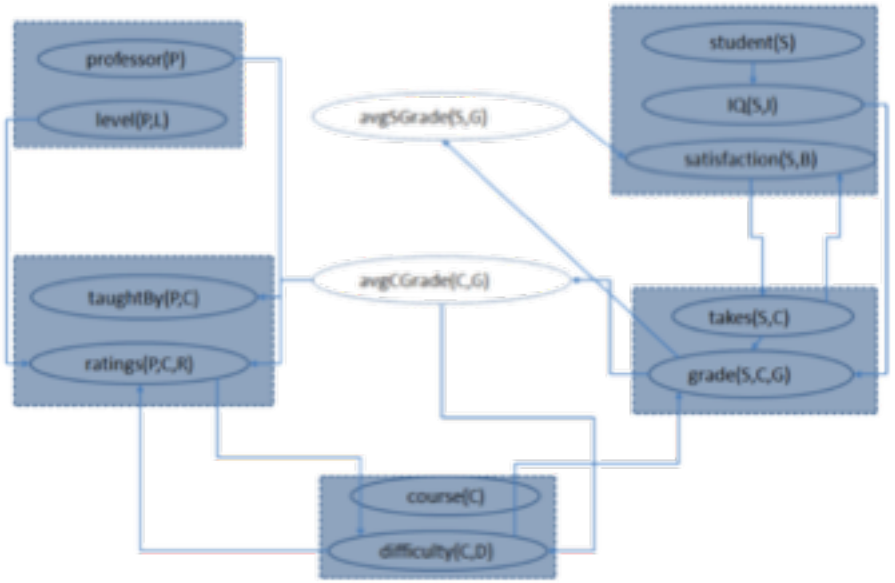
Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, Jude Shavlik: Gradient-based Boosting for Statistical Relational Learning: The Relational Dependency Network Case. In *Machine Learning*, 2012.

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

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

$P\left(a_k | parents(A_k)\right) \propto \psi^k$ ▷ 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 $P\left(y_k^i = 1; x_k^i\right), x_k^i = parents\left(y_k^i\right)$ ▷ Probab. 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

FOIL: extend example

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

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.

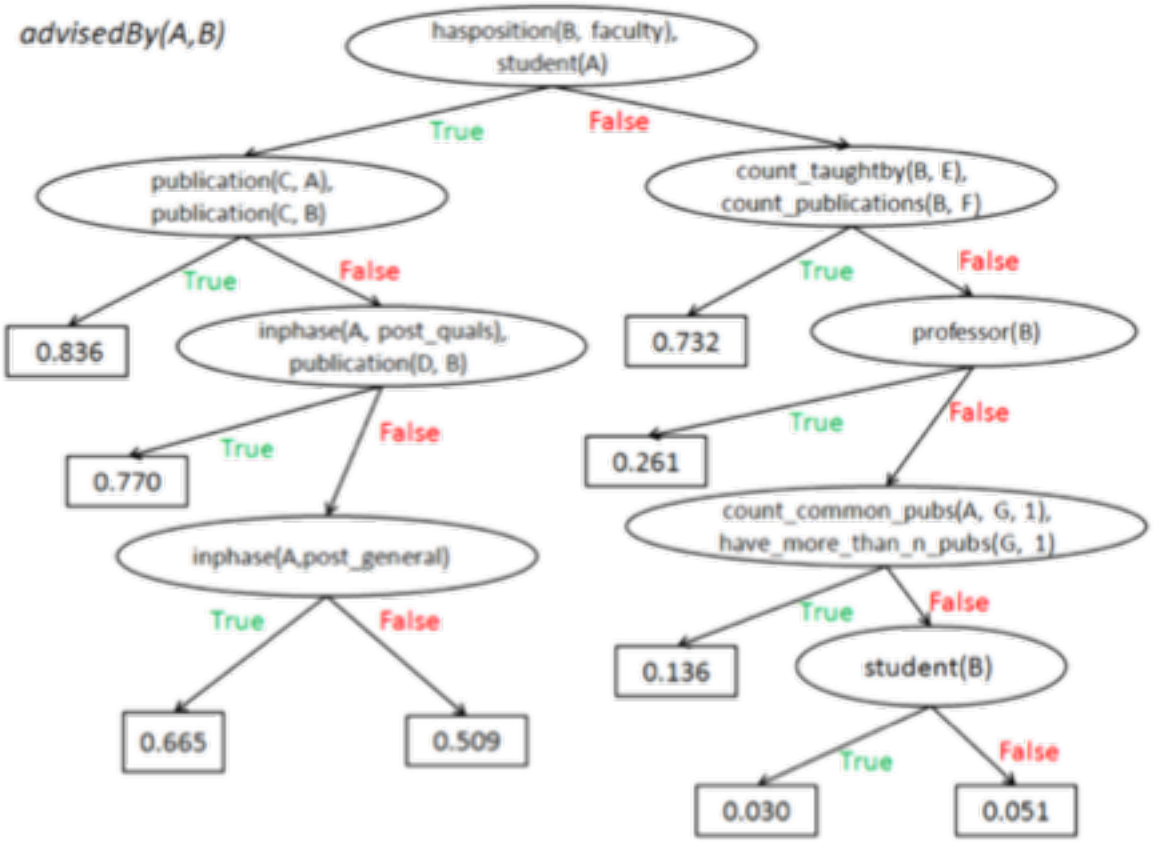
Bagging

- 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 parameterised models or MLNs

Outline: 4. Lifted Learning

A. *Recap: Learning directed models*
   - Maximum likelihood (ML) estimation, expectation-maximisation (EM), structural EM

B. *Learning undirected models*
   - Iterative fitting procedure

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

D. *Statistical relational learning*
   - First-order inductive learning
   - Decision tree representations

⇒ Next: Approximate Inference