RL in Robotics & Probabilistic Inference

Machine Learning B

708.062 08W 1sst KU

Dr. Stefan Häusler

Institut für Grundlagen der Informationsverarbeitung

Technische Universität Graz

Course WS 2010/11

Third homework set

Reinforcement learning in robotics:

- Policy gradient methods
- Reward weighted regression

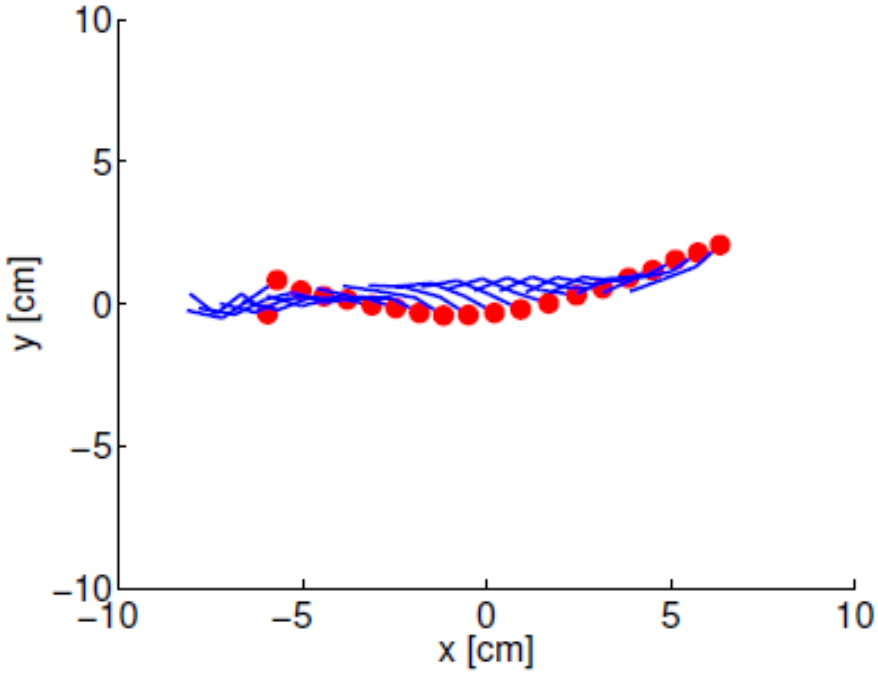
Approximate inference in Bayesian networks:

- Conditional independence in Bayesian networks
- Gibbs sampling

Policy Gradient Methods

Policy Search with a simple swimming robot:

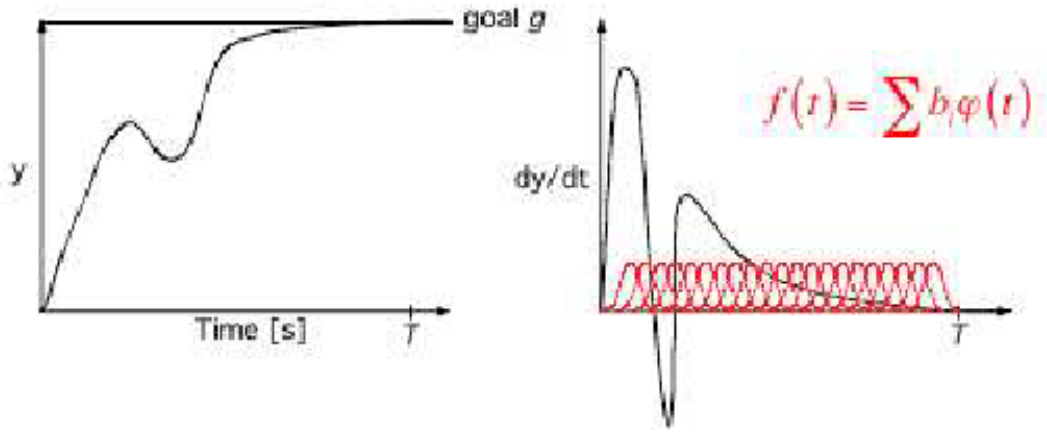
- The Policy models the swimming behavior
- The task is to swim as fast as possible using a minimum amount of energy (torques)
- Given a Matlab framework your task is to implement and evaluate different Policy Gradient algorithms.



Example behavior of a 3 link swimmer

The swimming behavior representation

The Movement is represented as Dynamic Movement Primitive.



$$\dot{z}_i = -\alpha(g_i - y_i) - \beta z_i, \qquad \dot{y}_i = z_i + f_i(\ldots)$$

$$f_i(x) = \frac{\sum_j G_j(x) b_{ij}}{\sum_j G_j(x)}$$

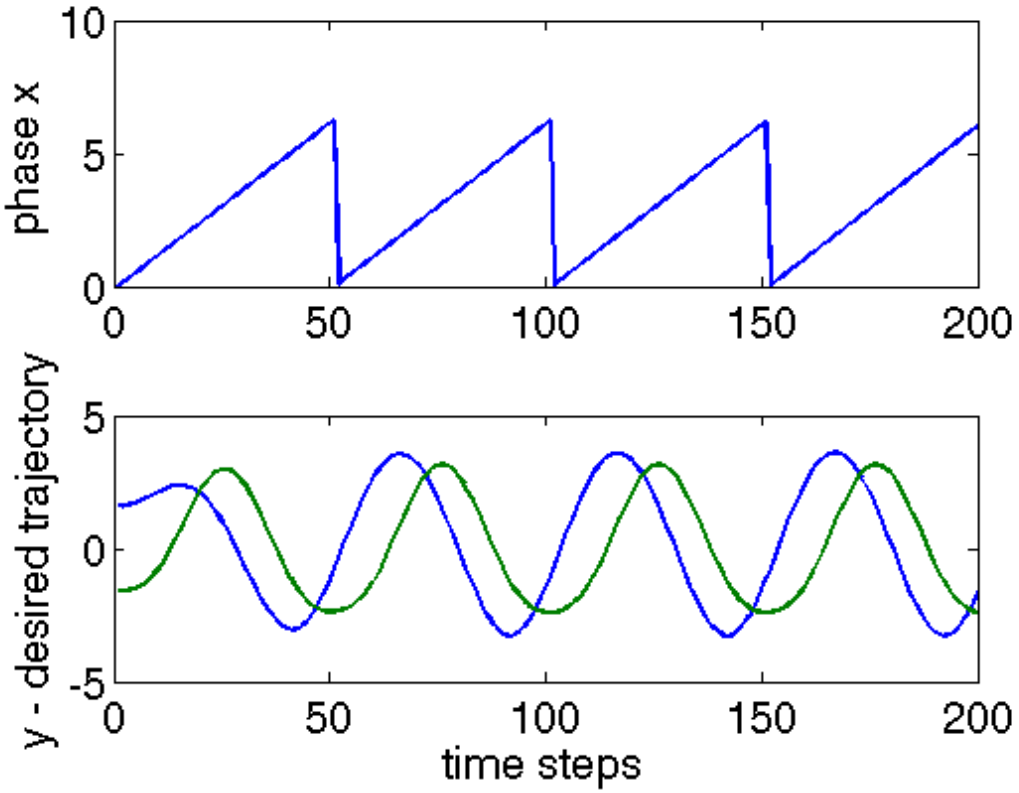
We need to learn the b's

The swimming behavior representation

The Movement is represented as Dynamic Movement Primitive.

We now want to represent a rhythmic movement:

- Periodic phase signal
- But actually we don't care: Already implemented in the framework.



Policy Gradient

Policy Gradient Methods:

$$J(\theta) = \int_{\tau \in \mathcal{T}} p_\theta(\tau) r(\tau) d\tau,$$

$$\theta_{h+1} = \theta_h + \alpha \nabla_\theta J(\theta_h)$$

3 Algorithms to compare:

- Finite Differences
- REINFORCE
- Policy Gradient Theorem

Algorithm 1 of 3: Finite Differences

Finite Difference Methods

Noise on policy parameters

- Apply I times small variations $\Delta\theta_i$ to the parameter vector θ
- Generate roll-outs (trajectories) to estimate $\Delta J_i \approx J(\theta + \Delta\theta_i) - J(\theta)$
- The gradient estimate can then be estimated by regression :

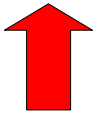
$$g_{FD} = (\Delta\Theta^T \Delta\Theta)^{-1} \Delta\Theta \Delta\mathbf{J},$$

with $\Delta\Theta = [\Delta\theta_1, \Delta\theta_2, \ldots, \Delta\theta_I]$ and
$\Delta\mathbf{J} = [\Delta J_1, \Delta J_2, \ldots, \Delta J_I]$

Likelihood ratio: We need a stochastic policy

How do we define a stochastic policy with DMPs?

$$\pi(\mathbf{a}|x, \mathbf{z}, \mathbf{y}, \mathbf{b}) = \mathcal{N}(\mathbf{a}|\dot{\mathbf{y}} = \mathbf{f_b}(x) + \mathbf{z}, \sigma^2\mathbf{I})$$



Noise on actions

The actions are the desired joint velocities y'

Algorithm 2 of 3: REINFORCE

REINFORCE

$$\mathbf{g}_{RF} = \left\langle \nabla_\theta \log p(\tau) \left(\sum_{l=0}^{T-1} \gamma^l r_l \right) \right\rangle,$$

$$\mathbf{g}_{RF} = \left\langle \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi(\mathbf{a}_t | \mathbf{s}_t; \theta) \right) \left(\sum_{l=0}^{T-1} \gamma^l r_l \right) \right\rangle,$$

where $\langle \dots \rangle$ denotes the expectation calculated by I rollouts.

Policy Gradient : How to calculate $\nabla_\theta \log \pi(\mathbf{a}|\mathbf{s};\theta)$

Example for DMPs: $\pi(\mathbf{a}|x,\mathbf{z},\mathbf{y},\mathbf{b}) = \mathcal{N}(\mathbf{a}|\mathbf{\Phi}^T(x)\mathbf{b}+\mathbf{z},\sigma^2\mathbf{I})$ Take the log...

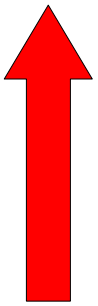
$$\log \pi(\mathbf{a}|x,\mathbf{z},\mathbf{y},\mathbf{b}) = -\frac{1}{2\sigma^2}(\mathbf{a}-(\mathbf{\Phi}^T(x)\mathbf{b}+\mathbf{z}))^T(\mathbf{a}-(\mathbf{\Phi}^T(x)\mathbf{b}+\mathbf{z}))+\text{const}$$

Take the derivative...

$$\nabla_\mathbf{b} \log \pi(\mathbf{a}|x,\mathbf{z},\mathbf{y},\mathbf{b}) = \frac{1}{\sigma^2}(\mathbf{a}-(\mathbf{\Phi}^T(x)\mathbf{b}+\mathbf{z}))^T\mathbf{\Phi} = \frac{1}{\sigma^2}\boldsymbol{\epsilon}\mathbf{\Phi},$$

with ϵ being the noise term $(\mathbf{a} = \mu + \boldsymbol{\epsilon},\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\sigma^2\mathbf{I}))$, .

- Measures correlation between features $\mathbf{\Phi}$ and noise $\boldsymbol{\epsilon}$



All these quantities are given by the Matlab framework

Algorithm 3 of 3: Policy Gradient Theorem

Policy Gradient Theorem (Sutton *et al.*, 1999)

Intuition : Actions in the future do not influence rewards in the past

$$\mathbf{g}_{PG} = \left\langle \sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi(\mathbf{a}_t|\mathbf{s}_t;\theta) \left(Q^\pi(\mathbf{s}_t, \mathbf{a_t}) - b \right) \right\rangle,$$

with $Q^t(\mathbf{s}_t, \mathbf{a_t}) = \left\langle \sum_{l=t}^{T-1} \gamma^{l-t} r_l \right\rangle$

Matlab Framework

```
clear all;
close all;
addpath([pwd filesep 'model']);

E = initE();

b = [3.2928   2.5650 -0.2796 -3.1864 -2.7956   1.8124...
    -1.7125   3.7744   3.0170 -1.3655 -2.6099 -0.5691];

sigma = 0;
[R, epsilon, Phi, rewards, s, a, y, dy] = E.J(b, sigma);
plotPolicy(E, b);
```

Policy Parameters

Stochasticity of the Policy

Reward Weighted Regression: Cannon Warfare

Learn how to shoot a cannon ball to different target positions under varying wind conditions

Parametrized Policy :

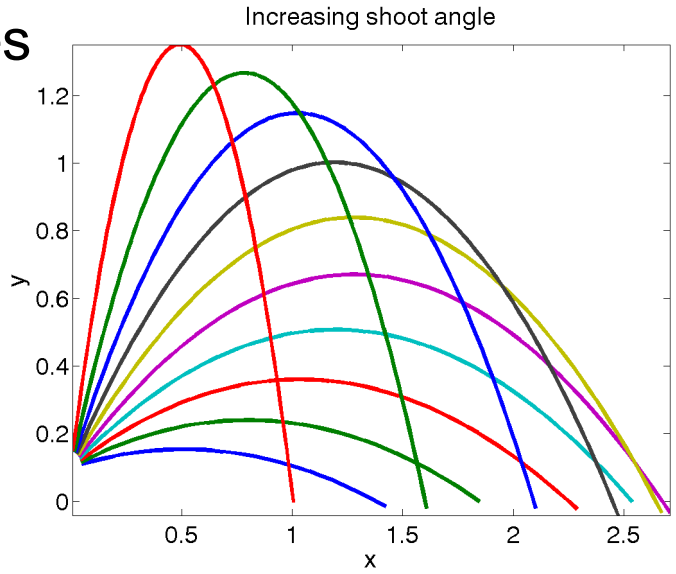
- Initial Angle α
- Initial Velocity v

Learn to choose different parameters in different initial conditions

- Target Distance x_T, Wind Strength w_S
- Hierarchical Policy : $\pi_\beta(\alpha, v | x_T, w_S)$

Reward Weighted Regression: Cannon Warfare

Illustration with different shoot angles



Use a stochastic Gaussian Policy $\mathcal{N}([\alpha, v]|\Phi(x_T, w_S)\beta, \Sigma)$

- Phi : Feature Vector (normalized RBF-network, 10x10)
- Beta : Parameter Vector, you need to learn this!
- Sigma : used for exploration (constant)

RWR: Formulas

Reward-Weighted Regression

Given :

- Inputdata $\Phi = [\phi_1; \phi_2; \ldots; \phi_I]$
- Target-vectors $\Theta = [\theta_1; \theta_2; \ldots; \theta_I]$
- Reward-weighting $\mathbf{W} = \mathrm{diag}([r_1, r_2, \ldots, r_I])$

In our case matrix $[\alpha_1 v_1; \ldots; \alpha_I v_I]$

Calculate reward-weighted least-squares solution

$$\beta = (\Phi^T \mathbf{W} \Phi + \lambda \mathbf{I})^{-1} \Phi^T \mathbf{W} \Theta$$

- $\lambda \ldots$ regularization parameter (ridge regression)

RWR: Cannon Warfare

Learning Procedure : Initialize empty training set

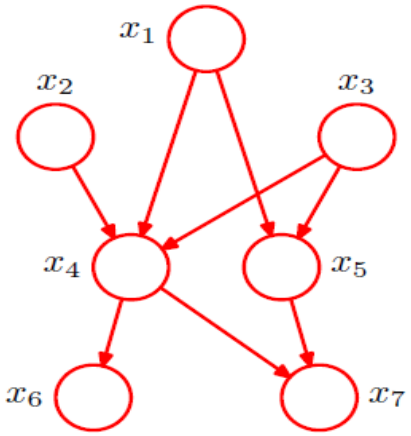
Repeat 500 times

- Generate 10 new samples
  - Choose initial state [x_T, w_S] randomly
  - Calculate α, v using current policy
  - Generate rollout (shoot), calculate reward r
  - Add data to your training set
- Re-estimate beta with RWR (using full training set)
- Evaluate policy

Directed Graphical Models

Probabilistic graphical models offer

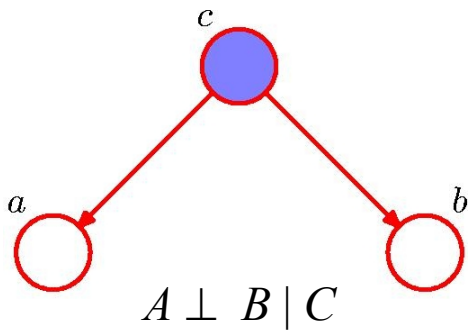
1. Simple way to **visualize structure** of a probabilistic model
2. Simple analysis of model properties, e.g. **conditional independence**
3. I**nference and learning** can be expressed in terms of graphical manipulations



$$p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$$

d(irected)-separation

The d-separation criterion ascertains whether a particular conditional independence statement for arbitrary non-intersecting sets of nodes *A, B* and *C* is implied by a given directed acyclic graph..



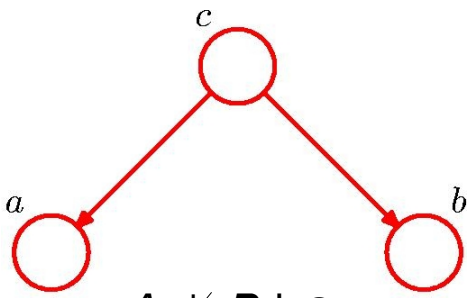
A is d-separated from *B* by *C* ⇔ *A* is conditional independent of *B* given *C*

A is d-separated from *B* by *C* ⇔ **All paths between *A* and *B* are blocked**

d-separation: case 1

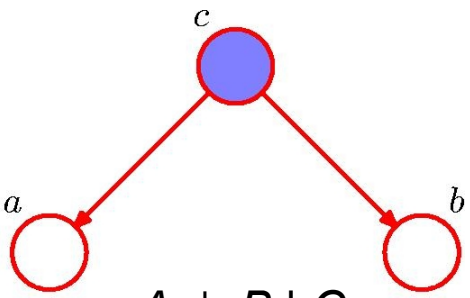
A **path is blocked** if it includes a node that is **tail-to-tail** and the node is in the set *C*.

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



$A \not\perp B \mid \varnothing$

$$P(A,B)=\sum_C P(A|C)P(B|C)P(C)$$
$$\neq P'(A)P'(B)$$



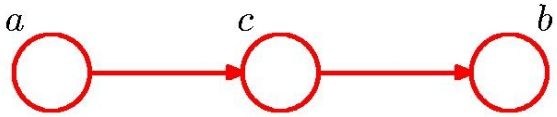
$A \perp B \mid C$

$$P(A,B|C)=\frac{P(A,B,C)}{P(C)}$$
$$=P(A|C)P(B|C)$$

d-separation: case 2

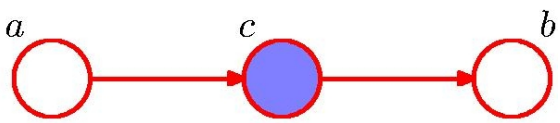
A **path is blocked** if it includes a node that is **head-to-tail** and the node is in the set *C*.

$$P(A,B,C)=P(B|C)\,P(C|A)\,P(A)$$



$A \not\perp B \mid \varnothing$

$$\begin{aligned} P(A,B) &= \sum_C P(A,B,C) \\ &= P(A)\sum_C P(B|C)\,P(C|A) \\ &\neq P'(A)\,P'(B) \end{aligned}$$



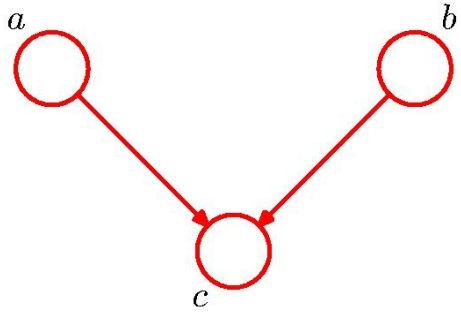
$A \perp B \mid C$

$$\begin{aligned} P(A,B|C) &= \frac{P(A,B,C)}{P(C)} \\ &= \frac{P(C|A)\,P(A)}{P(C)}\,P(B|C) \\ &= P(A|C)\,P(B|C) \end{aligned}$$

d-separation: case 3

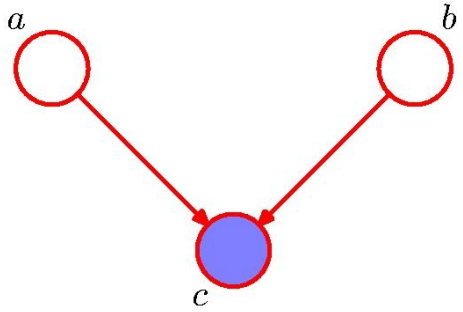
A **path is blocked** if it includes a node such that the arrows on the path meet **head-to-head** and neither the node nor its descendants are in the set *C*.

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



$A \perp B \mid \varnothing$

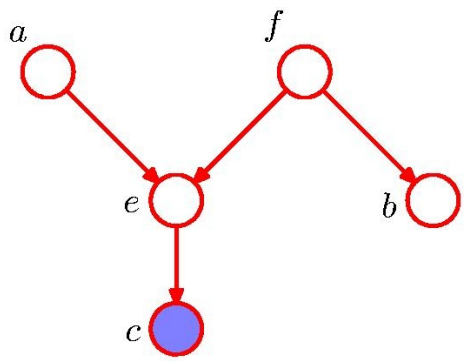
$$P(A,B) = \sum_C P(A,B,C) = P(A)P(B)$$



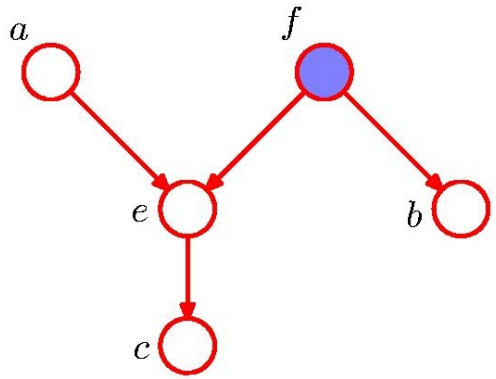
$A \not\perp B \mid C$

$$P(A,B|C) = \frac{P(A,B,C)}{P(C)} = \frac{P(A)P(B)P(C|A,B)}{P(C)} \neq P'(A|C)P'(B|C)$$

Examples: d-separation



(a)

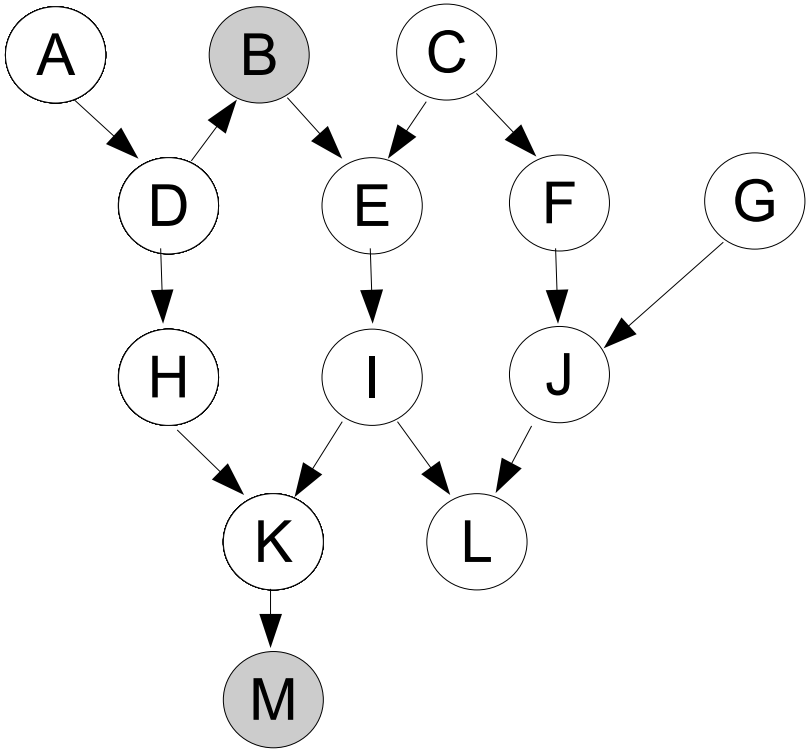


(b)

a) Path from *A* to *B* is neither blocked by *F* nor by *E*.

b) Path from *A* to *B* is blocked by *F* and by *E*.

Examples: d-separation



$A \perp G \mid B,M$?

Assignment 11

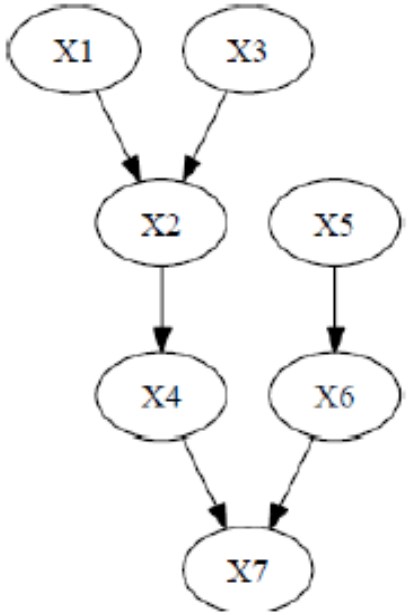


Figure 3: Bayesian network.

a) Determine whether the following conditional independence statements hold for the Bayesian network illustrated in Fig. 3. ...

Assignment 11

b) In your local nuclear power plant station, there is an alarm that senses when a temperature gauge exceeds a given threshold. The gauge measures the temperature of the core. Consider the Boolean variables A (alarm sounds), F_A (alarm is faulty), and F_G (gauge is faulty) and the multivalued nodes G (gauge reading) and T (actual core temperature).

1. Draw a Bayesian network for this domain, given that ...

2. 3. Give the conditional probability table associated with ...

Approximate Inference in Bayesian Networks

Given: $p(x_1, \ldots, x_K)$

Inference: $p(x_1, \mathbf{e}) = \sum_{v_2,\ldots,v_m} p(x_1, v_2, \ldots, v_m, \mathbf{e})$

e … evidence

Basic idea:

1) Draw N samples from a sampling distribution S
2) Compute an approximate posterior probability $\hat{P}$

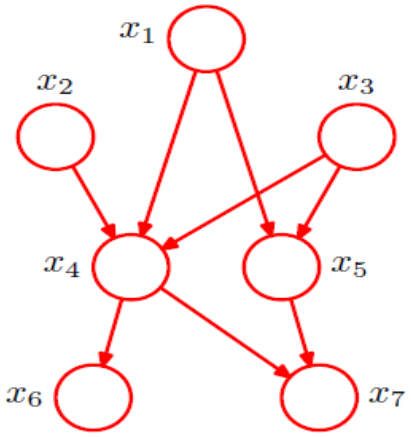
Sampling without evidence: Ancestral sampling

Basic idea

Sample a value for each variable in topological order using the specified conditional probabilities

function PRIOR-SAMPLE(bn) **returns** an event sampled from bn
  inputs: bn, a belief network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $\mathbf{x} \leftarrow$ an event with n elements
  for $i = 1$ **to** n **do**
    $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
      given the values of $Parents(X_i)$ in $\mathbf{x}$
  return $\mathbf{x}$

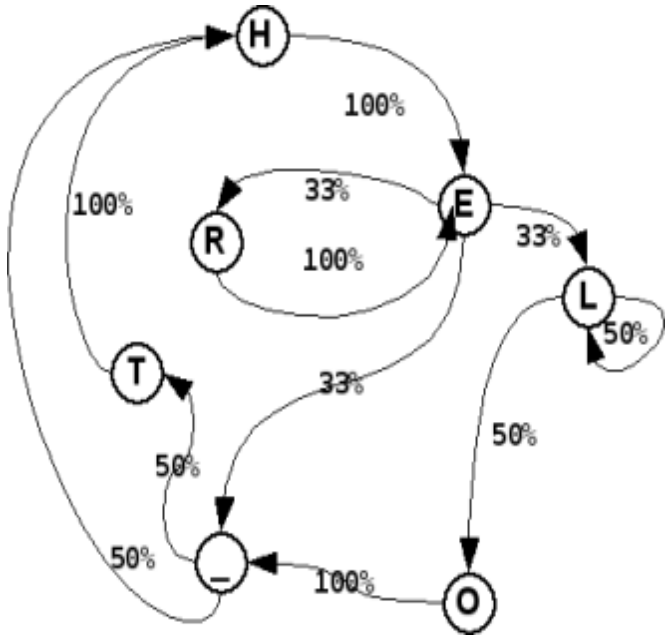


Sampling with evidence: Markov chain Monte Carlo methods

One designs a Markov chain, whose states s are all possible assignments of values $v_1, \ldots, v_K$ to the random variables $x_1, \ldots, x_K$ and whose stationary distribution p(s) defined by

$$p(\mathbf{s}) = \sum_{\mathbf{s}'} p_{MC}(\mathbf{s}' \rightarrow \mathbf{s}) \cdot p(\mathbf{s}')$$

is the distribution for which one wants to carry out probabilistic inference.



Entering evidence **e** amounts to a restriction of The dynamics of the Markov chain to a subgraph.

Then probabilistic inference $P(x_1|e)$ can be reduced to observing the dynamics of this Markov chain.

Simple MCMC sampling: Gibbs Sampling

Consider the distribution $p(\boldsymbol{x}) = p(x_1, \ldots, x_M)$

Gibbs Sampling

1. Initialize $\{x_i : i = 1, \ldots, M\}$
2. For $\tau = 1, \ldots, T$:
   - Sample $x_i^{(\tau+1)} \sim p(x_i | x_2^{(\tau)}, x_3^{(\tau)}, \ldots, x_M^{(\tau)})$.

i ... is chosen randomly from {1,... ,M}

Important issues

1) For Gibbs sampling the condition prob. distributions can be obtained by

$$p(x_j|x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_n) = \frac{p(x_1,\ldots,x_n)}{p(x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_n)} \propto p(x_1,\ldots,x_n)$$

2) Only factors of the joint distribution involving x_j have to be calculated.

Calculate for each value of x_j the product of these factors, normalize these values (one for each value of x_j) to obtain a correct prob. distr. and draw x_j according to this distribution.

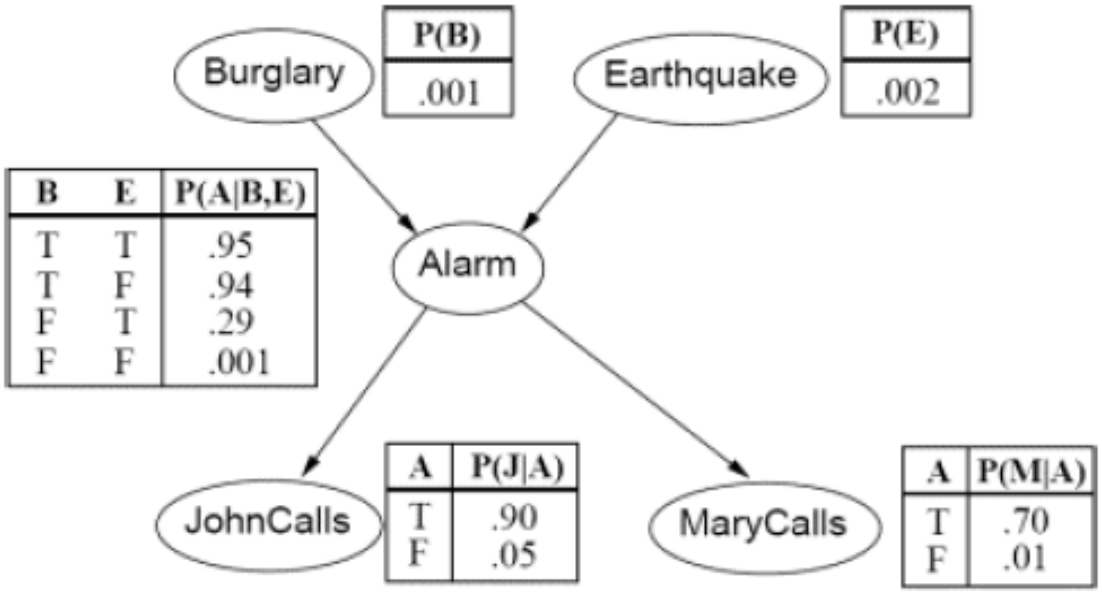
3) Burn in time:
Consecutive samples in the Markov chain are not independent. The burn in time is the number of steps between two samples to make them independent (steps required to forget initial conditions of the sampler)

Assignment 10

10 Approximate inference in Bayesian networks [3 P]

Apply Gibbs sampling to carry out approximate inference in Bayesian networks. You should estimate the (marginal) probability distribution of several variables in a Bayesian network, given the settings of a subset of the other variables (evidence). Implement the Gibbs algorithm in MATLAB based on the code provided `Gibbs.zip`[7] and test it on the three Bayesian networks shown below. Your code should run Gibbs sampling a specified number of iterations in order to estimate the required probability distributions. Since Gibbs sampling is a randomized, iterative algorithm, the actual number of iterations needed to estimate probabilities is an important issue, that will be explored as part of this assignment.

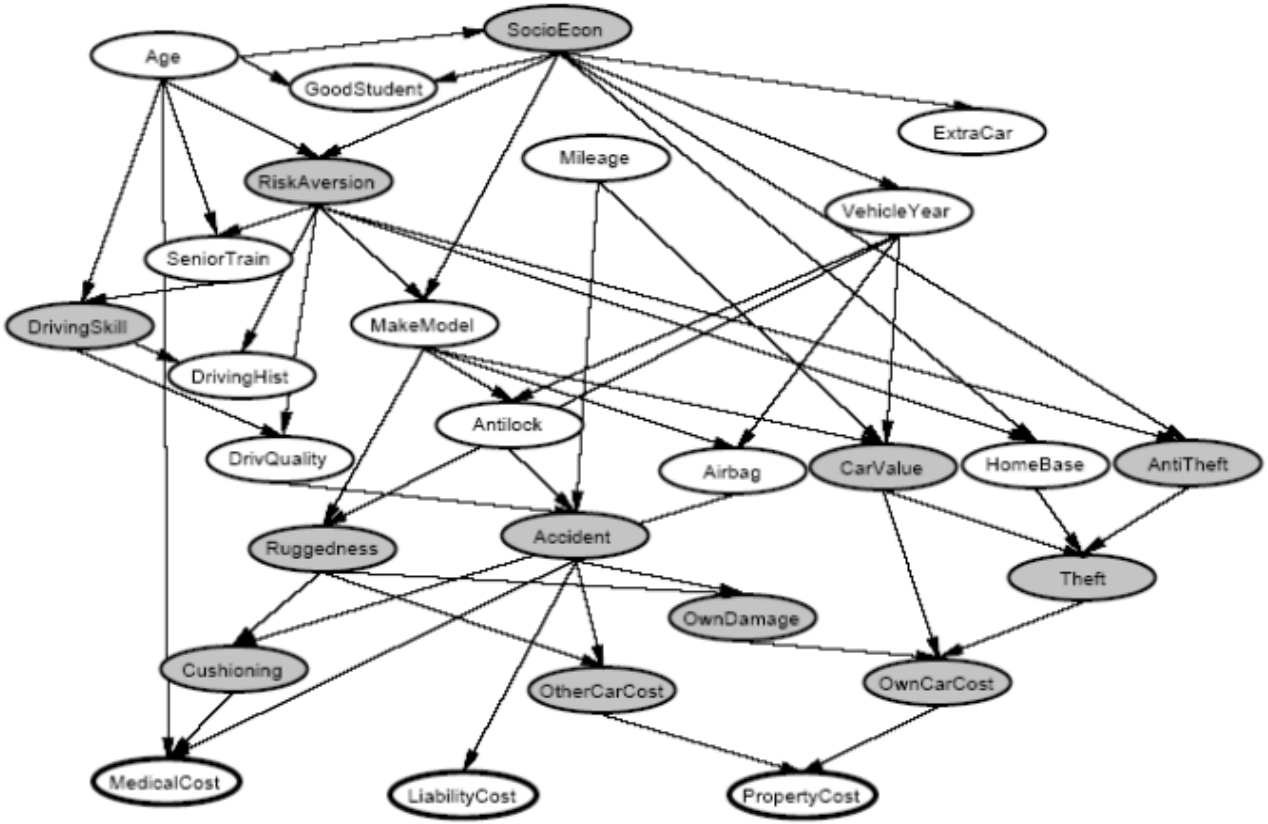
Verify explaining away



Explaining away:
If John and Mary call the observation of an earthquake has an effect on the probability of a burglary (variables are not independent anymore)

Estimate the burn-in time

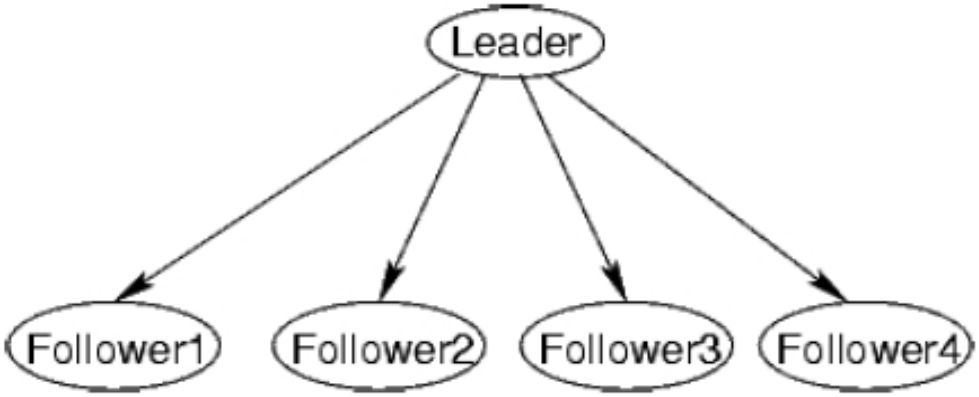
Insurance domain:



the network attempts to relate variables like the age and driving history of the individual to the cost and likelihood of property damage or bodily injury.

Explore the convergence properties of Gibbs sampling

Theoretically, Gibbs sampling will converge to the correct stationary probability distribution asymptotically, that is, as the number of iterations becomes very large. However, in practice, it is not always clear how many iterations actually suffice, as this example demonstrates.



The leader selects a bit at random (0 or 1 with equal probability), and then all of the followers choose their own bits, each one copying the leader's bit with 90% probability, or choosing the opposite of the leader's bit with 10% probability. Analyze leader-follower networks with k = 5, 10, 15, 20 followers have been provided in the files.

Data format for the Bayesian networks

Variables:

var(1).name: 'Leader'

var(1).nvalues: 2

var(1).Values: {'0' '1'}

Probability factors:

p(1).of: 1

p(1).cond_on: [2 3]

p(1).p: [2x2 double]

Indexing of probability tables:

p.p(index of the value variable 12, index of the value variable 1, …
index of the value variable 3)

Complete the MATLAB code

```
for ns = 1:nsamples
   % choose a variable to resample (not the evidence)
...
   % find factors where this variable appears
...
    % calc prob of all factor for each value of the selected variable
 ...
   % normalize the obtained prob values f
   nf = f/sum(f);

   % draw a value for the selected random variable
   nvidx = find(rand < cumsum(nf),1);
   x(...) = nvidx;

   % store the state
   X(:,ns) = x;
end
```