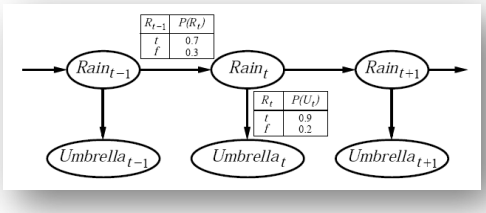
Artificial Intelligence 2

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Summary of last lecture

We know how to do **probabilistic reasoning over time**

- transition model $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$, sensor mode $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$
- Markov assumptions



Basic **inference tasks**:

- filtering: $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$
- prediction: $\mathbf{P}(\mathbf{X}_{t+k} \mid \mathbf{e}_{1:t})$ pro $k>0$
- smoothing: $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$ pro $k: 0 \leq k < t$
- most likely explanation: $\text{argmax}_{\mathbf{x}_{1:t}} \mathbf{P}(\mathbf{x}_{1:t} \mid \mathbf{e}_{1:t})$

Hidden Markov Models (HMM)

- a special case with a single state variable and a single sensor variable
- inference tasks solved by means of matrix operations

Localization (an example of HMM)

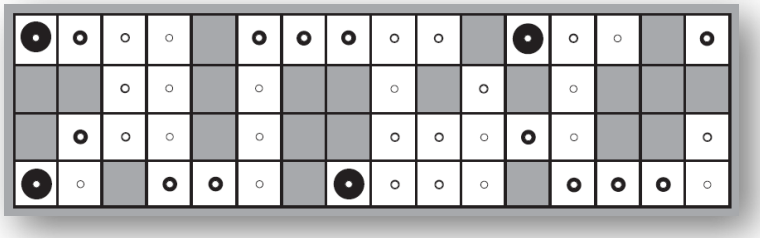
Assume a robot that moves randomly in a grid world, has a map of the world and (noisy) sensors reporting obstacles laying immediately to the north, south, east, and west. The robot needs to find its location.

A possible model:

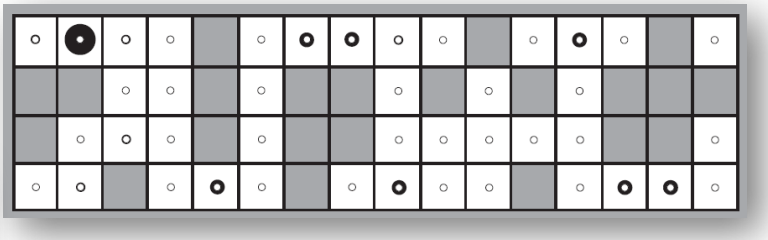
- **random variables** X_t describe robot's location at times t
  - possible values are 1,..,n for n locations
  - Nb(i) – a set of neighboring locations for location i
- **transition tables**
  - $P(X_{t+1}=j|X_t=i) = 1/ |Nb(i)|$, if $j \in Nb(i)$, otherwise 0
- **sensor variables** E_t describe observations (evidence) at times t (four sensor for four directions NSEW)
  - values indicate detection of obstacle at a given direction NSEW (16 values for all directions)
  - assume that sensor's error rate is ε
- **sensor tables**
  - $P(E_t=e_t|X_t=i) = (1-\varepsilon)^{4-d_{it}} \varepsilon^{d_{it}}$
    where d_{it} is the number of deviations of observation e_t from the true values for square i

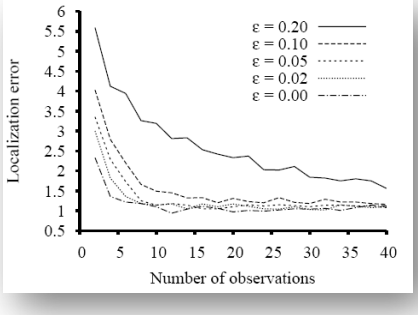
Localization (a practical example)

$\mathbf{P}(X_1 | E_1=NSW)$



$\mathbf{P}(X_2 | E_1=NSW, E_2=NS)$





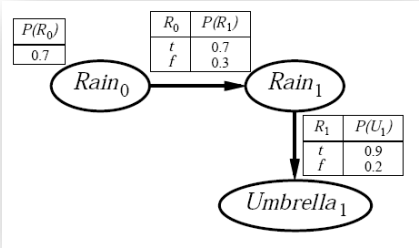
The localization error defined as the Manhattan distance from the true location

Dynamic Bayesian network (DBN) is a Bayesian network that represents a temporal probability model.

the variables and links exactly replicated from slice to slice

It is enough to describe one slice.

- prior distribution $\mathbf{P}(\mathbf{X}_0)$
- transition model $\mathbf{P}(\mathbf{X}_1 \mid \mathbf{X}_0)$
- sensor model $\mathbf{P}(\mathbf{E}_1 \mid \mathbf{X}_1)$

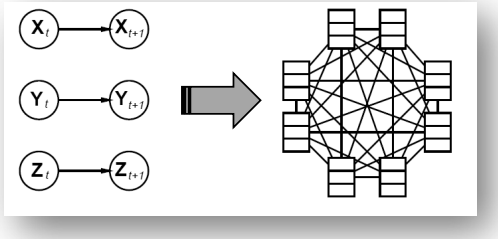


Each variable has parents either at the same slice or in the previous slice (Markov assumption).

A hidden Markov model is a special case of a dynamic Bayesian network.

Similarly, a dynamic Bayesian network can be encoded as a hidden Markov model

one random variable in HMM whose values are n-tuples of values of state variables in DBN



What is the difference?

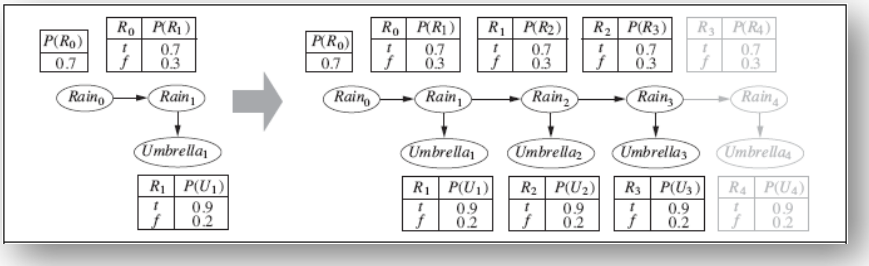
The relationship between DBN and HMM is roughly analogous to the relationship between ordinary Bayesian networks and full tabulated joint distribution.

- DBN with 20 Boolean state variables, each of which has three parents
  - the transition model has $20 \times 2^3 = 160$ probabilities
- Hidden Markov model has one random variable with 2^{20} values
  - the transition model has $2^{20} \times 2^{20} \approx 10^{12}$ probabilities
  - HMM requires much more space and inference is much more expensive

Exact inference in DBNs

Dynamic Bayesian networks are Bayesian networks and we already have algorithms for inference in Bayesian networks.

We can construct the full Bayesian network representation of a DBN by replicating slices to accommodate the observations (**unrolling**).



A naive application of unrolling would not be particularly efficient as the inference time will increase with new observations.

We can use an incremental approach that keeps in memory only two slices (via summing out the variables from previous slices).

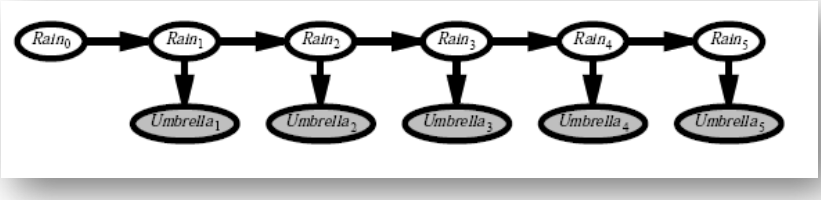
- This is similar to **variable elimination** in Bayesian networks.
- The bad news are that "constant" space to represent the **largest factor will be exponential** in the number of state variables.
  $O(d^{n+k})$, where n is the number of variables, d is the domain size, k is the maximum number of parents of any state variable

Approximate inference in DBNs

We can try approximate inference methods (**likelihood weighting** is most easily adapted to the DBN context).

We sample non-evidence nodes of the network in topological order, weighting each sample by the likelihood in accords to the observed evidence variables.

- each sample must go through a full network
- we can simply run all N samples together through the network (N samples are similar to the forward message, the samples themselves are approximate representation of the current state distribution)



There is a **problem**!

Samples are generated completely independently of the evidence!

- This behaviors decreases accuracy of the method as the weight of samples is usually very small and we need much larger number of samples
  - We need to **increase the number of samples exponentially** with t.
  - If a constant number of samples is used them the method accuracy is goes down significantly.

Particle filtering

We need to focus the set of samples on the high-probability regions of the sample space.

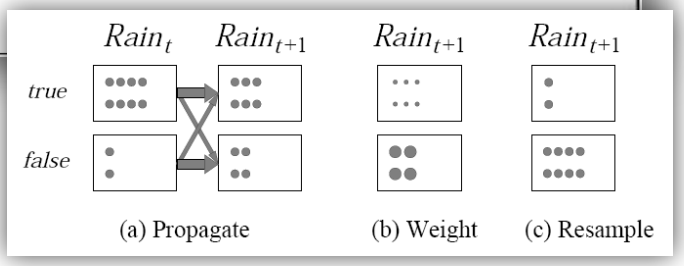
Particle filtering is doing exactly that by resampling the population of samples.

- a population of N samples is created by **sampling from the prior distribution $\mathbf{P}(\mathbf{X}_0)$**
- each sample is **propagated forward** by sampling the next state value $\mathbf{x}_{t+1}$ given the current value $\mathbf{x}_t$ for the sample, based on the transition model $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t)$
- each sample is **weighted** by the likelihood it assigns to the new evidence, $\mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1})$
- the population is **resampled** to generate a new population of N samples
  - each sample is selected from the current population such that the probability of selection is proportional to its weight (the new samples are unweighted)

Particle filtering algorithm

function PARTICLE-FILTERING(e,N, dbn) returns a set of samples for the next time step
  inputs: e, the new incoming evidence
    N, the number of samples to be maintained
    dbn, a DBN with prior $\mathbf{P}(\mathbf{X}_0)$, transition model $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$, and sensor model $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$
  static: S, a vector of samples of size N, initially generated from $\mathbf{P}(\mathbf{X}_0)$
  local variables: W, a vector of weights of size N

  for i = 1 to N **do**
    $S[i] \leftarrow$ sample from $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0 = S[\mathrm{i}])$
    $W[i] \leftarrow \mathbf{P}(\mathbf{e}|\mathbf{X}_1 = S[i])$
  $S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S, W)
  return S



Particle filtering (soundness)

Assume that samples at time t are consistent with probability distribution

$N(\mathbf{x}_t \mid \mathbf{e}_{1:t}) / N = P(\mathbf{x}_t \mid \mathbf{e}_{1:t})$

After **propagation** to time t+1, the number of samples is

$N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) = \Sigma_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t})$

The total **weight** of the samples in state $\mathbf{x}_{t+1}$ after evidence is

$W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t})$

After **resampling** we will get

$N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) / N = \alpha\, W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1})$

$= \alpha\, P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t})$

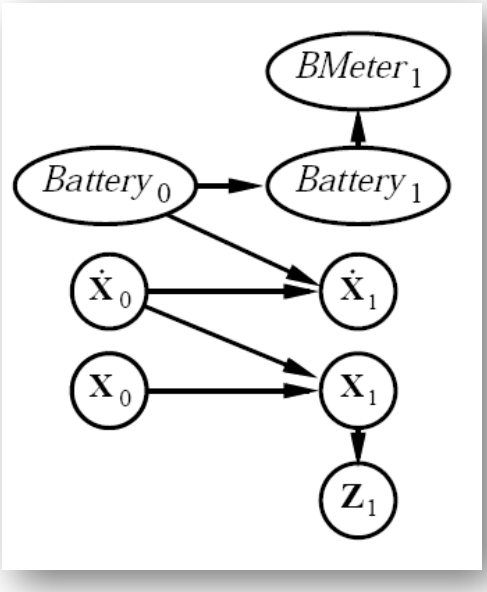
$= \alpha\, P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \Sigma_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t})$

$= \alpha'\, P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \Sigma_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t})$

$= P(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1})$

Complex example (DBN)

Let us consider a battery-powered mobile robot and model its behavior using a DBN.

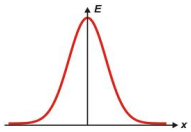


- we need state variables modelling **position** of the robot $\mathbf{X}_t = (X_t, Y_t)$, its **velocity** $\mathbf{X}_t = (X_t, Y_t)$, and actual **battery level**
  - the position at the next time step depends on the current position and velocity
  - the velocity at the next steps depends on the current velocity and the state of battery
- we assume some method of **measuring position** (a fixed camera or onboard GPS) $\mathbf{Z}_t$
- similarly we assume a sensor **measuring battery level** $\mathbf{BMeter}_t$

Sensor failures

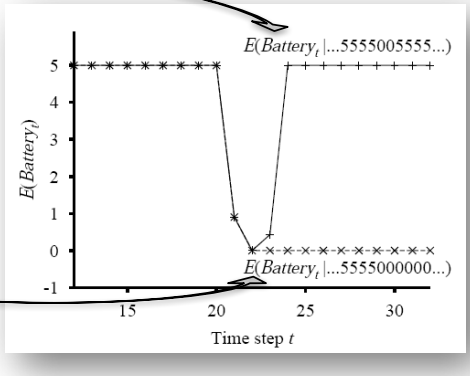
A fully accurate sensor uses a sensor model with probabilities 1.0 "along the diagonal" and probabilities 0.0 elsewhere.

In reality, noise always creeps into measurements. A Gaussian distribution with small variance might be used (for continuous measurements)
– **Gaussian error model**



Real sensors fail. When a sensor fails, it simply sends nonsense (but does not say that it that way). Then the Gaussian error model does not work as we need.

- after two wrong measurements we are almost certain that the battery is empty
- if the failure disappears then the battery level quickly returns to 5, as if by magic (**transient failures**)
  - in the meantime, the system may do some wrong decisions (shut down)
- if the wrong measures are still coming then the model is certain that the batters is empty (though a chance of such a sudden change is small) (**persistent failure**)



Sensor transient failure model

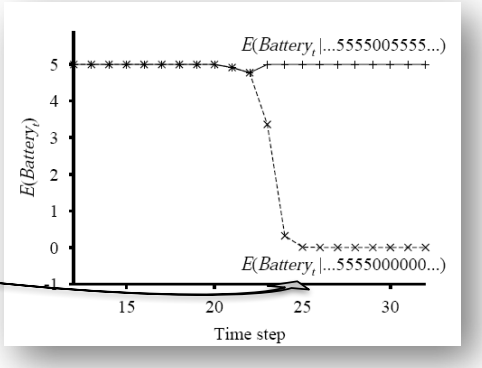
The simplest kind of failure model allows a certain probability that the sensor will return some completely incorrect value, regardless of the true state of the world.

$P(BMeter_t=0 \mid Battery_t=5) = 0.03$

Let us call this the **transient failure model**.

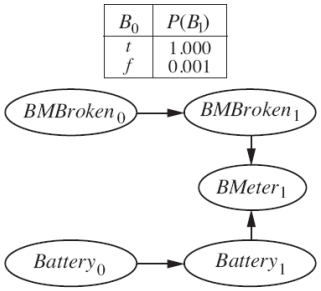
The model brings some "inertia" that helps to overcome temporary blips in the meter reading.

However, after more wrong readings the robot gradually coming to believe that the its battery is empty while in fact it may by that the meter has failed!

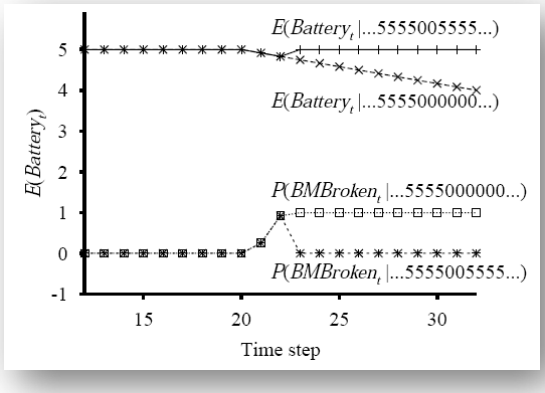


Sensor persistent failure model

To model persistent failures we use additional state variable, **BMBroken**, that describes the status of the battery meter.



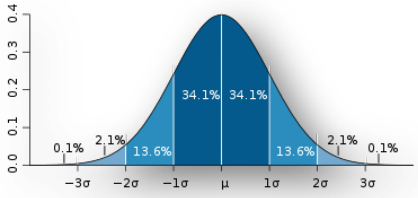
- The **persistence arc** has a CPT that gives a small probability of failure in any given time step (0.001), but specifies that the sensor stays broken once it breaks.
- When the sensor is OK, the sensor model is identical to the transient failure model.
- Once the sensor is known to be broken, the robot can only assume that its battery discharges at the "normal" rate.



Continuous variables

So far we assumed discrete random variables so the probability distribution can be captured by tables.

- **How can we handle continuous variables?**
  - **discretization**
    - dividing up the possible values into a fixed set of intervals
    - often results in a considerably loss of accuracy and very large CPTs
  - we can also use **standard families of probability density functions** that are specified by a finite number of parameters
    - Gaussian (or normal) distribution
      - described by the mean μ and the variance σ^2 as parameters



$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)},$$

Continuous variables (conditional probability)

How are the conditional probability tables specified in hybrid Bayesian networks?

- dependence of **continuous variable on the continuous variable** can be described using linear Gaussian distribution
  - » the mean values varies linearly with the value of the parent
  - » standard deviation is fixed

$$P(Cost = c|Harvest = h, Subsidy? = true) = N(a_t h + b_t, \sigma_t)(c) = \frac{1}{\sigma_t\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{c-(a_t h + b_t)}{\sigma_t}\right)^2\right)$$

- dependence of **continuous variable on the discrete variable**
  - » for each value of the discrete variable we specify parameters of standard distribution
- Dependence of **discrete variables on the continuous variable**
  - » "soft" threshold function

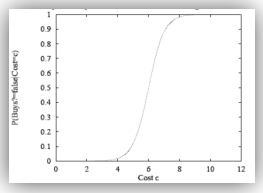
probit (probability unit) distribution

$$\Phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx$$
$$P(Buys? = true \mid Cost = c) = \Phi((-c+\mu)/\sigma)$$

logit (logistic function) distribution

$$P(Buys? = true \mid Cost = c) = \frac{1}{1 + exp(-2\frac{-c+\mu}{\sigma})}$$

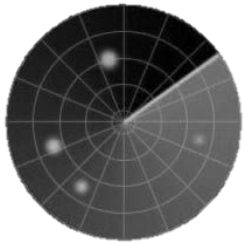
  - » probit is often a better fit to real situations, (the underlying decision process has a hard threshold, but the precise location of th threshold is subject to random Gaussian noise)
  - » logit has much longer "tails", sometimes easier to deal with mathematically

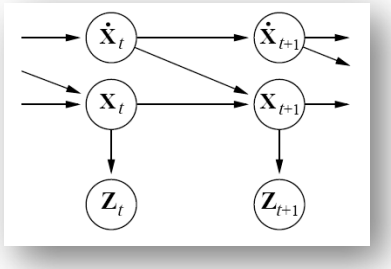


Kalman filters

Assume a problem of detecting actual position of an aircraft based on observations on radar screen. We will model the problem using a dynamic Bayesian network.

- random state variables describe actual **location** and **speed** of the aircraft
  - the next location depends on the previous location and speed and can be modelled using linear Gaussian distribution
    $P(X_{t+\Delta}=x_{t+\Delta} \mid X_t=x_t , \dot{X}_t=\dot{x}_t) = N(x_t+\dot{x}_t\Delta,\sigma^2)(x_{t+\Delta})$
- we observe location Z_t
  - again, we can use Gaussian distribution in the sensor model





Kalman filters (properties)

Gaussian distribution has some nice properties when solving the tasks of filtering, prediction, and smoothing.

- If the distribution $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ is Gaussian and the transition model $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t)$ is linear Gaussian then $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t})$ is also a Gaussian distribution.

$$\mathrm{P}(\mathrm{X}_{t+1}|\mathrm{e}_{1:t}) = \int_{\mathrm{x}_t} \mathrm{P}(\mathrm{X}_{t+1}|\mathrm{x}_t) P(\mathrm{x}_t|\mathrm{e}_{1:t})\, d\mathrm{x}_t$$

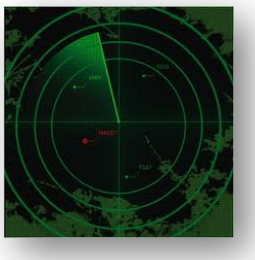
- If $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t})$ is Gaussian and the sensor model $\mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})$ is linear Gaussian then $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1})$ is also a Gaussian distribution.

$$\mathrm{P}(\mathrm{X}_{t+1}|\mathrm{e}_{1:t+1}) = \alpha \mathrm{P}(\mathrm{e}_{t+1}|\mathrm{X}_{t+1}) \mathrm{P}(\mathrm{X}_{t+1}|\mathrm{e}_{1:t})$$

- To solve tasks we can use the message passing technique.

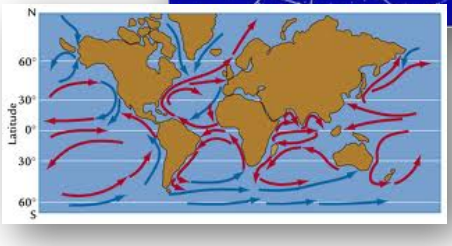
Kalman filters (applications)

- Classical application of the Kalman filter is **tracking movements** of aircrafts and missiles using radars.



- Kalman filters are used to **reconstruct trajectories** of particles in physics and monitoring ocean currents.

- The range of applications is much larger than tracking of motion: **any system characterized by continuous state variables and noisy measurements** will do (pulp mills, chemical plants, nuclear reactors, plant ecosystems, and national economies).



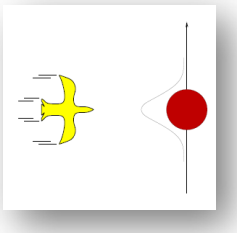
Kalman filters (non-linearities)

Kalman filters assumes linear Gaussian transition and sensor models.

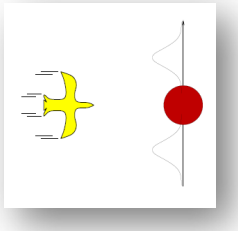
What if the model is non-linear?

Example: assume a bird heading at high speed straight for a tree trunk

Kalman filter



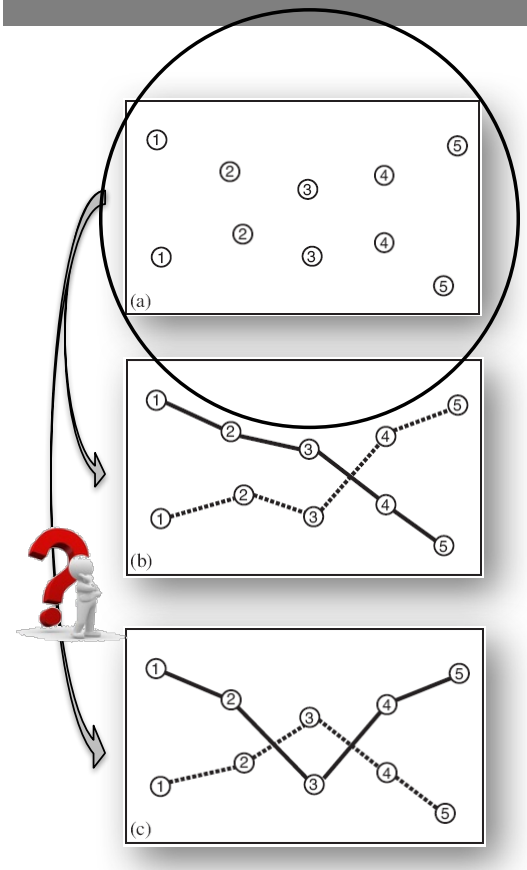
a more realistic model



A standard solution: **switching Kalman filter**

- **multiple Kalman filters** run in parallel, each using a different model of the system (one for straight flight, one for sharp left turns , and one for sharp right turns)
- a **weighted sum of predictions** is used, where the weight depends om how well each filter fits the current data
- this is a special case of DBN obtained by **adding a discrete "maneuver" state variable** to the network

Keeping track of many objects



We have considered state estimation problems involving a single object.

What do happen if **two or more objects** generate the observations?

Additional problem:

Uncertainty about which object generated which observation!

Keeping track of many objects (approaches)

Data association problem:
the problem of associating observation data with the objects that generated them

Exact reasoning means summing out the variables over all possible assignments of objects to observations

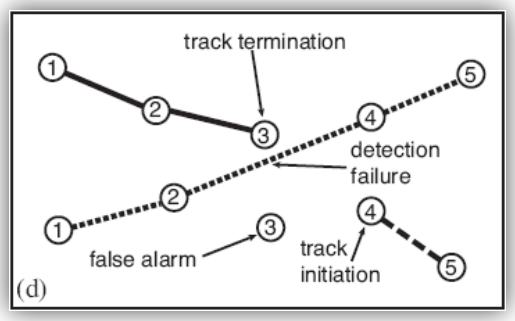
- for a single time slice and n objects it means n! mappings
- for T time slices we have $(n!)^T$ mappings

Many different **approximate methods** are used.

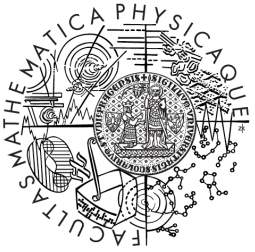
- choose a single "best" assignment at each time step
  - **nearest-neighbor filter** (chooses the closest pairing of predicted position and observation)
  - a better approach is to choose the assignment that **maximizes the joint probability** of the current observations given the predicted positions (the Hungarian algorithm)
- Any method that commits to as single best assignment at each time step fails miserably under more difficult conditions (the prediction on the next step may be significantly wrong)
  - **particle filtering** (maintains a large collection of possible current assignments)
  - **MCMC** (explores the space of possible current assignments)

Keeping track of many objects (in reality)

Real applications of data association are typically much more complicated.



- **false alarm (clutter)** (observations not caused by real objects)
- **detection failures** (no observation is reported for a real object)
- **new and disappearing objects** (new objects arrive and old ones disappear)



Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz