CS 188: Artificial Intelligence

Review of Utility, MDPs, RL, Bayes' nets

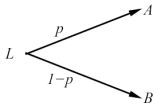
DISCLAIMER: It is insufficient to simply study these slides, they are merely meant as a quick refresher of the high-level ideas covered. You need to study all materials covered in lecture, section, assignments and projects !

Pieter Abbeel – UC Berkeley

Many slides adapted from Dan Klein

Preferences

- An agent must have preferences among:
  - Prizes: A, B, etc.
  - Lotteries: situations with uncertain prizes



$$L = [p, A;\ (1-p), B]$$

- Notation:

$A \succ B$	A preferred over B
$A \sim B$	indifference between A and B
$A \succeq B$	B not preferred over A

Rational Preferences

- Preferences of a rational agent must obey constraints.
  - The axioms of rationality:

Orderability
$(A \succ B) \vee (B \succ A) \vee (A \sim B)$

Transitivity
$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

Continuity
$A \succ B \succ C \Rightarrow \exists p\ [p, A;\ 1-p, C] \sim B$

Substitutability
$A \sim B \Rightarrow [p, A;\ 1-p, C] \sim [p, B; 1-p, C]$

Monotonicity
$A \succ B \Rightarrow$
$(p \geq q \Leftrightarrow [p, A;\ 1-p, B] \succeq [q, A;\ 1-q, B])$

- Theorem: Rational preferences imply behavior describable as maximization of expected utility

MEU Principle

- Theorem:
  - [Ramsey, 1931; von Neumann & Morgenstern, 1944]
  - Given any preferences satisfying these constraints, there exists a real-valued function U such that:

$$U(A) \geq U(B) \ \Leftrightarrow\ A \succeq B$$

$$U([p_1, S_1;\ \ldots\ ;\ p_n, S_n]) = \textstyle\sum_i p_i U(S_i)$$

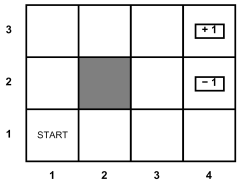
- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tictactoe, reflex vacuum cleaner

Recap MDPs and RL

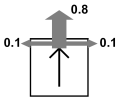
- Markov Decision Processes (MDPs)
  - Formalism (S, A, T, R, gamma)
  - Solution: policy pi which describes action for each state
  - Value Iteration (vs. Expectimax --- VI more efficient through dynamic programming)
  - Policy Evaluation and Policy Iteration
- Reinforcement Learning (don't know T and R)
  - Model-based Learning: estimate T and R first
  - Model-free Learning: learn without estimating T or R
    - Direct Evaluation [performs policy evaluation]
    - Temporal Difference Learning [performs policy evaluation]
    - Q-Learning [learns optimal state-action value function Q*]
    - Policy Search [learns optimal policy from subset of all policies]
  - Exploration
- Function approximation --- generalization

Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s,a,s')
    - Prob that a from s leads to s'
    - i.e., P(s' | s,a)
    - Also called the model
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state



- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don't know the transition or reward functions



What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means:

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$
$$=$$
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- Can make this happen by proper choice of state space

Value Iteration

- Idea:
  - $V_i^*(s)$: the expected discounted sum of rewards accumulated when starting from state s and acting optimally for a horizon of i time steps.
- Value iteration:
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for horizon i+1:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

  - This is called a value update or Bellman update
  - Repeat until convergence
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do
  - At convergence, we have found the optimal value function V* for the discounted infinite horizon problem, which satisfies the Bellman equations: $\forall s \in S: \quad V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$

8

Complete Procedure

- 1. Run value iteration (off-line)
  - This results in finding V*
- 2. Agent acts. At time t the agent is in state s_t and takes the action a_t:

$$\arg\max_a \sum_{s'} T(s_t, a, s')[R(s_t, a, s') + \gamma V^*(s')]$$

9

Policy Iteration

- Policy evaluation: with fixed current policy π, find values with simplified Bellman updates:
  - Iterate for i = 0, 1, 2, … until values converge

$$\forall s: \quad V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_k}(s')\right]$$

- Will converge (policy will not change) and resulting policy optimal

10

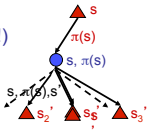
Sample-Based Policy Evaluation?

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')]$$

- Who needs T and R? Approximate the expectation with samples (drawn from T!)

$$sample_1 = R(s, \pi(s), s_1') + \gamma V_i^{\pi}(s_1')$$
$$sample_2 = R(s, \pi(s), s_2') + \gamma V_i^{\pi}(s_2')$$
$$\ldots$$
$$sample_k = R(s, \pi(s), s_k') + \gamma V_i^{\pi}(s_k')$$

$$V_{i+1}^{\pi}(s) \leftarrow \frac{1}{k} \sum_i sample_i$$

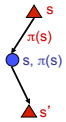


Almost! (i) Will only be in state s once and then land in s' hence have only one sample → have to keep all samples around? (ii) Where do we get value for s'?

11

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience (s,a,s',r)
  - Likely s' will contribute updates more often
- Temporal difference learning
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!



Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

12

Exponential Moving Average

- Exponential moving average
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Forgets about the past (distant past values were wrong anyway)
  - Easy to compute from the running average

$$\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

Detour: Q-Value Iteration

- Value iteration: find successive approx optimal values
  - Start with $V_0(s) = 0$, which we know is right (why?)
  - Given V_i, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

- But Q-values are more useful!
  - Start with $Q_0(s,a) = 0$, which we know is right (why?)
  - Given Q_i, calculate the q-values for all q-states for depth i+1:

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

Q-Learning

- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your new sample estimate:

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough but not decrease it too quickly!
- Neat property: off-policy learning
  - learn optimal policy without following it

Exploration Functions

- Simplest: random actions (ε greedy)
  - Every time step, flip a coin
  - With probability ε, act randomly
  - With probability 1-ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
- Exploration functions
  - Explore areas whose badness is not (yet) established
  - Take a value estimate and a count, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$ (exact form not important)

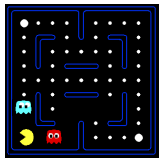
$$Q_{i+1}(s,a) \leftarrow (1-\alpha)Q_i(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right)$$

now becomes:

$$Q_{i+1}(s,a) \leftarrow (1-\alpha)Q_i(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} f(Q_i(s',a'), N(s',a'))\right)$$

Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Feature Functions

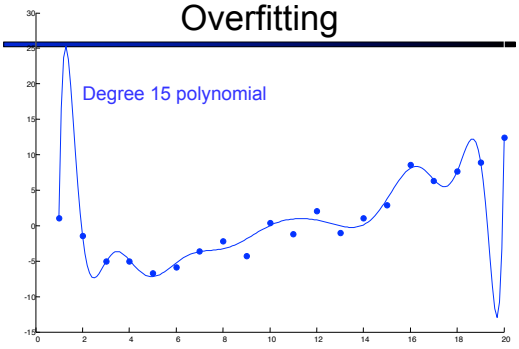
- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \dots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but be very different in value!

Overfitting



19

Policy Search

- Problem: often the feature-based policies that work well aren't the ones that approximate V / Q best
- Solution: learn the policy that maximizes rewards rather than the value that predicts rewards
- This is the idea behind policy search, such as what controlled the upside-down helicopter
- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical

20

Probability recap

- Conditional probability $P(x|y) = \frac{P(x,y)}{P(y)}$
- Product rule $P(x,y) = P(x|y)P(y)$
- Chain rule $P(X_1, X_2, \dots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots$
- X, Y independent iff: $\forall x, y : P(x,y) = P(x)P(y)$
  equivalently, iff: $\forall x, y \ : \ P(x|y) = P(x)$
  equivalently, iff: $\forall x, y \ : \ P(y|x) = P(y)$
- X and Y are conditionally independent given Z iff:
  $\forall x, y, z : P(x,y|z) = P(x|z)P(y|z)$
  equivalently, iff: $\forall x, y, z \ : \ P(x|y,z) = P(x|z)$
  equivalently, iff: $\forall x, y, z \ : \ P(y|x,z) = P(y|z)$

21

Inference by Enumeration

- P(sun)?
- P(sun | winter)?
- P(sun | winter, hot)?

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

22

Bayes' Nets Recap

- Representation
  - Chain rule -> Bayes' net = DAG + CPTs
- Conditional Independences
  - D-separation
- Probabilistic Inference
  - Enumeration (exact, exponential complexity)
  - Variable elimination (exact, worst-case exponential complexity, often better)
  - Probabilistic inference is NP-complete
  - Sampling (approximate)

23

Chain Rule → Bayes net

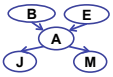
- Chain rule: can ***always*** write ***any*** joint distribution as an incremental product of conditional distributions

$$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

$$P(x_1, x_2, \dots x_n) = \prod_i P(x_i|x_1 \dots x_{i-1})$$

- Bayes nets: make conditional independence assumptions of the form:

$$P(x_i|x_1 \cdots x_{i-1}) = P(x_i|parents(X_i))$$

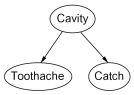


giving us:

$$P(x_1, x_2, \dots x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$$

24

Probabilities in BNs



- Bayes' nets implicitly encode joint distributions
  - As a product of local conditional distributions
  - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:

$$P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i | parents(X_i))$$

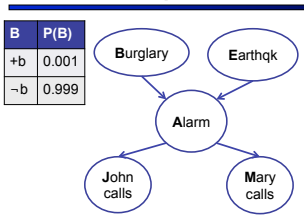
  - Example:

$$P(+cavity, +catch, \neg toothache)$$

- This lets us reconstruct any entry of the full joint
- Not every BN can represent every joint distribution
  - The topology enforces certain conditional independencies

25

Example: Alarm Network



B	P(B)
+b	0.001
¬b	0.999

E	P(E)
+e	0.002
¬e	0.998

A	J	P(J|A)
+a	+j	0.9
+a	¬j	0.1
¬a	+j	0.05
¬a	¬j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	¬m	0.3
¬a	+m	0.01
¬a	¬m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	¬a	0.05
+b	¬e	+a	0.94
+b	¬e	¬a	0.06
¬b	+e	+a	0.29
¬b	+e	¬a	0.71
¬b	¬e	+a	0.001
¬b	¬e	¬a	0.999

Size of a Bayes' Net for $P(X_1, X_2, \ldots X_n)$

- How big is a joint distribution over N Boolean variables?
  2^N
- Size of representation if we use the chain rule
  2^N
- How big is an N-node net if nodes have up to k parents?
  $O(N * 2^{k+1})$
- Both give you the power to calculate
- BNs:
  - Huge space savings!
  - Easier to elicit local CPTs
  - Faster to answer queries

27

Bayes Nets: Assumptions

- Assumptions made by specifying the graph:

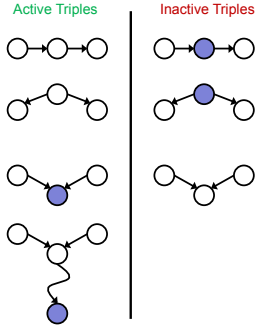
$$P(x_i | x_1 \cdots x_{i-1}) = P(x_i | parents(X_i))$$

- Given a Bayes net graph additional conditional independences can be read off directly from the graph
- Question: Are two nodes *guaranteed to be independent* given certain evidence?
- If no, can prove with a counter example
  - I.e., pick a set of CPT's, and show that the independence assumption is violated by the resulting distribution
- If yes, can prove with
  - Algebra (tedious)
  - D-separation (analyzes graph)

28

D-Separation

- Question: Are X and Y conditionally independent given evidence vars {Z}?
  - Yes, if X and Y "separated" by Z
  - Consider all (undirected) paths from X to Y
  - No active paths = independence!
- A path is active if each triple is active:
  - Causal chain A → B → C where B is unobserved (either direction)
  - Common cause A ← B → C where B is unobserved
  - Common effect (aka v-structure) A → B ← C where B *or one of its descendents* is observed
- All it takes to block a path is a single inactive segment



D-Separation

- Given query $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$?
- Shade all evidence nodes
- For all (undirected!) paths between and
  - Check whether path is active
    - If active return $X_i \not\perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$
- (If reaching this point all paths have been checked and shown inactive)
  - Return $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$

30

Example

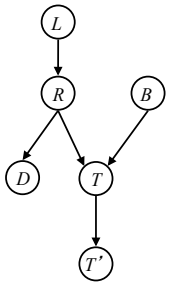
$L \perp\!\!\!\perp T' | T$ Yes

$L \perp\!\!\!\perp B$ Yes

$L \perp\!\!\!\perp B | T$

$L \perp\!\!\!\perp B | T'$

$L \perp\!\!\!\perp B | T, R$ Yes



31

All Conditional Independences

- Given a Bayes net structure, can run d-separation to build a complete list of conditional independences that are necessarily true of the form

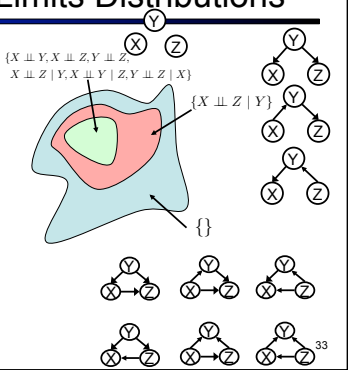
$$X_i \perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$$

- This list determines the set of probability distributions that can be represented by Bayes' nets with this graph structure

32

Topology Limits Distributions

- Given some graph topology G, only certain joint distributions can be encoded
- The graph structure guarantees certain (conditional) independences
- (There might be more independence)
- Adding arcs increases the set of distributions, but has several costs
- Full conditioning can encode any distribution

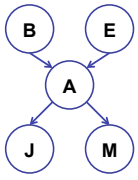


33

Inference by Enumeration

- Given unlimited time, inference in BNs is easy
- Recipe:
  - State the marginal probabilities you need
  - Figure out ALL the atomic probabilities you need
  - Calculate and combine them
- Example:

$$P(+b | +j, +m) = \frac{P(+b, +j, +m)}{P(+j, +m)}$$



34

Example: Enumeration

- In this simple method, we only need the BN to synthesize the joint entries

$$P(+b, +j, +m) =$$
$$P(+b)P(+e)P(+a|+b,+e)P(+j|+a)P(+m|+a) +$$
$$P(+b)P(+e)P(-a|+b,+e)P(+j|-a)P(+m|-a) +$$
$$P(+b)P(-e)P(+a|+b,-e)P(+j|+a)P(+m|+a) +$$
$$P(+b)P(-e)P(-a|+b,-e)P(+j|-a)P(+m|-a)$$

35

Variable Elimination

- Why is inference by enumeration so slow?
  - You join up the whole joint distribution before you sum out the hidden variables
  - You end up repeating a lot of work!
- Idea: interleave joining and marginalizing!
  - Called "Variable Elimination"
  - Still NP-hard, but usually much faster than inference by enumeration

36

Variable Elimination Outline

- Track objects called factors
- Initial factors are local CPTs (one per node)

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

- Any known values are selected
  - E.g. if we know $L = +\ell$, the initial factors are

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(+\ell|T)$

+t	+l	0.3
-t	+l	0.1

- VE: Alternately join factors and eliminate variables

Variable Elimination Example

$P(R)$

+r	0.1
-r	0.9

Join R

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(R,T)$

+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

Sum out R

$P(T)$

+t	0.17
-t	0.83

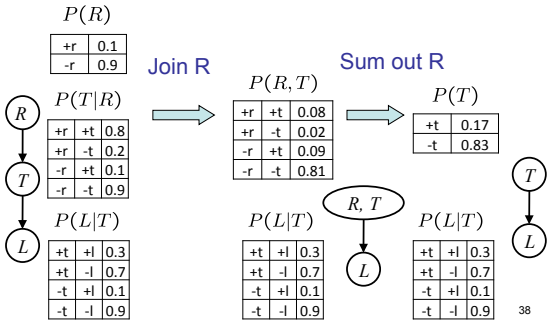
$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

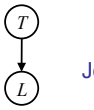
R, T

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9



Variable Elimination Example



Join T

T, L

Sum out T

L

$P(T)$

+t	0.17
-t	0.83

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

$P(T,L)$

+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

$P(L)$

+l	0.134
-l	0.886

* VE is variable elimination

Example

$P(B|j,m) \propto P(B,j,m)$

$P(B)$	$P(E)$	$P(A|B,E)$	$P(j|A)$	$P(m|A)$

Choose A

$P(A|B,E)$
$P(j|A)$
$P(m|A)$

$\times$ $P(j,m,A|B,E)$ $\sum$ $P(j,m|B,E)$

$P(B)$	$P(E)$	$P(j,m|B,E)$

Example

$P(B)$	$P(E)$	$P(j,m|B,E)$

Choose E

$P(E)$
$P(j,m|B,E)$

$\times$ $P(j,m,E|B)$ $\sum$ $P(j,m|B)$

$P(B)$	$P(j,m|B)$

Finish with B

$P(B)$
$P(j,m|B)$

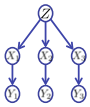
$\times$ $P(j,m,B)$ Normalize $P(B|j,m)$

General Variable Elimination

- Query: $P(Q|E_1 = e_1, \ldots E_k = e_k)$
- Start with initial factors:
  - Local CPTs (but instantiated by evidence)
- While there are still hidden variables (not Q or evidence):
  - Pick a hidden variable H
  - Join all factors mentioning H
  - Eliminate (sum out) H
- Join all remaining factors and normalize

Another (bit more abstractly worked out) Variable Elimination Example

Query: $P(X_3|Y_1 = y_1, Y_2 = y_2, Y_3 = y_3)$



Start by inserting evidence, which gives the following initial factors:

$$p(Z)p(X_1|Z)p(X_2|Z)p(X_3|Z)p(y_1|X_1)p(y_2|X_2)p(y_3|X_3)$$

Eliminate X_1, this introduces the factor $f_1(Z, y_1) = \sum_{x_1} p(x_1|Z)p(y_1|x_1)$, and:

$$p(Z)f_1(Z, y_1)p(X_2|Z)p(X_3|Z)p(y_2|X_2)p(y_3|X_3)$$

Eliminate X_2, this introduces the factor $f_2(Z, y_2) = \sum_{x_2} p(x_2|Z)p(y_2|x_2)$, and:

$$p(Z)f_1(Z, y_1)f_2(Z, y_2)p(X_3|Z)p(y_3|X_3)$$

Eliminate Z, this introduces the factor $f_3(y_1, y_2, y_3, X_3) = \sum_z p(z)f_1(z, y_1)f_2(z, y_2)p(X_3|z)p(y_3|X_3)$, and:

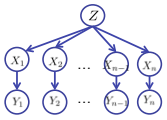
$$f_3(y_1, y_2, y_3, X_3)$$

Normalizing over X_3 gives $P(X_3|y_1, y_2, y_3)$.

Computational complexity critically depends on the largest factor being generated in this process. Size of factor = number of entries in table. In example above (assuming binary) all factors generated are of size 2 --- as they all only have one variable (Z, Z, and X3 respectively).

Variable Elimination Ordering

- For the query $P(X_n|y_1,\ldots,y_n)$ work through the following two different orderings as done in previous slide: Z, X_1, …, X_{n-1} and X_1, …, X_{n-1}, Z. What is the size of the maximum factor generated for each of the orderings?



- Answer: 2^n versus 2 (assuming binary)
- In general: the ordering can greatly affect efficiency.

Computational and Space Complexity of Variable Elimination

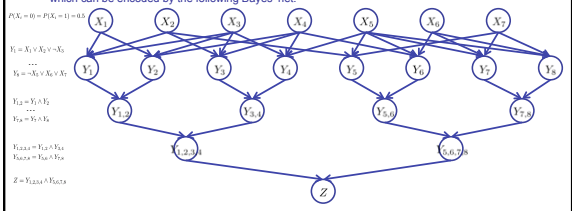
- The computational and space complexity of variable elimination is determined by the largest factor
- The elimination ordering can greatly affect the size of the largest factor.
  - E.g., previous slide's example 2^n vs. 2
- Does there always exist an ordering that only results in small factors?
  - No!

Worst Case Complexity?

- Consider the 3-SAT clause:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4 \vee \neg x_5) \wedge (x_2 \vee x_5 \vee x_7) \wedge (x_4 \vee x_5 \vee x_6) \wedge (\neg x_5 \vee x_6 \vee \neg x_7) \wedge (\neg x_5 \vee \neg x_6 \vee x_7)$$

  which can be encoded by the following Bayes' net:



- If we can answer P(z) equal to zero or not, we answered whether the 3-SAT problem has a solution.
- Subtlety: why the cascaded version of the AND rather than feeding all OR clauses into a single AND? Answer: a single AND would have an exponentially large CPT, whereas with representation above the Bayes' net has small CPTs only.
- Hence inference in Bayes' nets is NP-hard. No known efficient probabilistic inference in general.

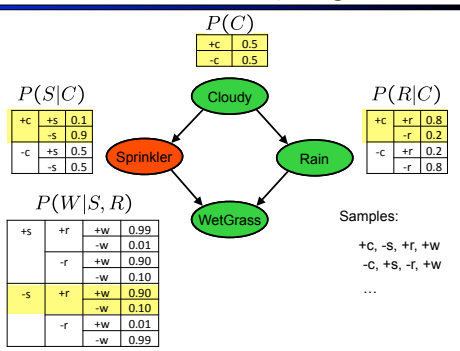
Polytrees

- A polytree is a directed graph with no undirected cycles
- For poly-trees you can always find an ordering that is efficient
  - Try it!!
- Cut-set conditioning for Bayes' net inference
  - Choose set of variables such that if removed only a polytree remains
  - Think about how the specifics would work out!

Approximate Inference: Sampling

- Basic idea:
  - Draw N samples from a sampling distribution S
  - Compute an approximate posterior probability
  - Show this converges to the true probability P
- Why? Faster than computing the exact answer
- Prior sampling:
  - Sample ALL variables in topological order as this can be done quickly
- Rejection sampling for query $P(Q|E_1 = e_1, \ldots E_k = e_k)$
  - = like prior sampling, but reject when a variable is sampled inconsistent with the query, in this case when a variable E_i is sampled differently from e_i
- Likelihood weighting for query $P(Q|E_1 = e_1, \ldots E_k = e_k)$
  - = like prior sampling but variables E_i are not sampled, when it's their turn, they get set to e_i, and the sample gets weighted by $P(e_i \mid$ value of parents(e_i) in current sample)
- Gibbs sampling: repeatedly samples each non-evidence variable conditioned on all other variables → can incorporate downstream evidence

Prior Sampling



Samples:

+c, -s, +r, +w

-c, +s, -r, +w

…

Example

- We'll get a bunch of samples from the BN:

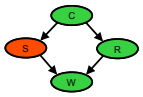
  +c, -s, +r, +w

  +c, +s, +r, +w

  -c, +s, +r, -w

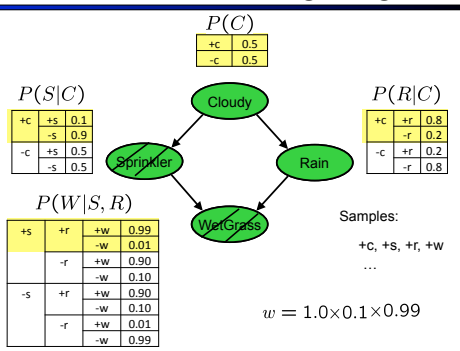
  +c, -s, +r, +w

  -c, -s, -r, +w



- If we want to know P(W)
  - We have counts <+w:4, -w:1>
  - Normalize to get P(W) = <+w:0.8, -w:0.2>
  - This will get closer to the true distribution with more samples
  - Can estimate anything else, too
  - What about P(C| +w)? P(C| +r, +w)? P(C| -r, -w)?
  - Fast: can use fewer samples if less time

Likelihood Weighting



Samples:

+c, +s, +r, +w

…

$w = 1.0 \times 0.1 \times 0.99$

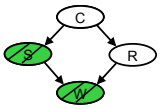
Likelihood Weighting

- Sampling distribution if z sampled and e fixed evidence

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | \text{Parents}(Z_i))$$

- Now, samples have weights

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | \text{Parents}(E_i))$$



- Together, weighted sampling distribution is consistent

$$S_{\text{WS}}(z, e) \cdot w(z, e) = \prod_{i=1}^{l} P(z_i | \text{Parents}(z_i)) \prod_{i=1}^{m} P(e_i | \text{Parents}(e_i))$$

$$= P(\mathbf{z}, \mathbf{e})$$

Gibbs Sampling

- *Idea*: instead of sampling from scratch, create samples that are each like the last one.
- *Procedure*: resample one variable at a time, conditioned on all the rest, but keep evidence fixed.
- *Properties*: Now samples are not independent (in fact they're nearly identical), but sample averages are still consistent estimators!
- *What's the point*: both upstream and downstream variables condition on evidence.