

- 1 intro
- 2 uninformed search
- 3 informed search
- 4 constraint satisfaction
- 5 local search
- 6 uncertainty
- 7 bayesian networks
- 8 bayesian networks
- 9 decision theory
- 10 decision networks
- 11 probabilistic reasoning over time
- 12 markov decision process
- 13 decision tree learning
- 14 statistical learning
- 15 neural networks
- 16 deep neural networks
- 17 reinforced learning
- 18 deep reinforced learning
- 19 deep reinforced learning
- 20 multi-armed bandits
- 21 sum product networks

MODULE 1 – INTRO

A **rational agent** chooses whichever action that maximizes the **expected** value of its performance measure given the **percept sequence to date**.

PEAS → Performance measure, Environment, Actuators, Sensors
(specify the task environment)

Properties of TASK ENVIRONMENT

- Fully observable vs partionally observable
- Deterministic vs stochastic
- Episodic vs sequential
- Static vs dynamic
- Discrete vs continuous
- Single agent vs multi agent

MODULE 2 – UNINFORMED SEARCH

Common characteristic:

- Fully observable
- Deterministic (state doesn't change unless there is an action)
- Sequential (one action depends on previous one and determines future ones)
- Static environment
- Discrete
- Single agent

We use **Search Trees** (which are constructed 'on the fly' by algorithm)

Uninformed search methods expand nodes based on "distance" from start node (never look ahead to the goal)

Search Node

- State
- Parent node and operator applied to parent to reach current node
- Cost of path so far
- Depth of node

Generic Search Algorithm

1. Initialize search algorithm with initial state of problem
2. Repeat
 1. If no candidate nodes can be expanded, return failure
 2. Choose leaf node for expansion, according to **search strategy**
 3. If node contains goal state, return solution
 4. Otherwise, expand the node by applying legal operators to the state within the node and add resulting nodes to the tree

******we use priority queues to keep track of nodes that need to be expanded

b *branching factor (max number of edges that come out of any node)*

d *depth of shallowest goal node*

m *maximum length of any path in the state space*

complete – algorithm guaranteed to find solution (if it exists)

optimal – algorithm finds optimal solution

time complexity

space complexity

BREADTH-FIRST SEARCH

- Nodes expanded by level
- Uses FIFO queue
- Complete (if b is finite), optimal, time = $O(b^d)$, space = $O(b^d)$

UNIFORM COST SEARCH

- Variation of BFS, but instead of expanding shallowest node, it expands the node with the lowest path cost
- Optimal, complete if $\epsilon > 0$ (ϵ is minimum action cost), time = $O(b^{\lceil C^*/\epsilon \rceil})$, space = $O(b^{\lceil C^*/\epsilon \rceil})$
(C^* is cost of optimal solution)

DEPTH-FIRST SEARCH

- Uses LIFO queue (stack)
- Not complete, not optimal, time = $O(b^m)$, space = $O(bm)$

DEPTH-LIMITED SEARCH

- Use a depth limit l
- Not complete, not optimal, time = $O(b^l)$, space = $O(bl)$

ITERATIVE-DEEPENING SEARCH

- Does DFS but with that limit that increases every time
- The last term (last level) dominates
- Good when there is a large state space and the maximum depth of the solution is unknown
- Complete, optimal, time = $O(b^d)$, space = $O(bd)$

MODULE 3 – INFORMED SEARCH

In Informed search, knowledge is often based on the (merit) value of being at a node

Heuristic function, $h(n)$, *guesses* the cost of reaching the goal from node n

- 'domain specific'
- $h(n) = 0$ when n is goal node
- if $h(n_1) < h(n_2)$, then we guess that it is cheaper to reach the goal from n_1 than it is from n_2

GREEDY BEST-FIRST SEARCH

- node with lowest h -value is expanded
- doesn't take into account the cost of the path so far
- not optimal, not complete (can get stuck in loops – unless theres a repetition check)
- space = exponential in worst case, time = $O(b^m)$ where m is the max depth of a tree

A* SEARCH

$$f(n) = g(n) + h(n)$$

$g(n)$ is the cost of the path to node n

$h(n)$ is the heuristic estimate

$h^*(n)$ denotes the true minimal cost to the goal from node n

- expand node in the queue with lowest f -value
- along the path, f will keep increasing and eventually get to solution cost
- terminates only when the goal state is popped from the queue
- not optimal with thw wrong heuristic (non admissible heuristics)
- optimal if heuristic is admissible **admissible** $\rightarrow h(n) \leq h^*(n)$
- complete if heuristic is **consistent**
- **monocity (consistent)** $\rightarrow h(n) \leq \text{cost}(n, n') + h(n')$
 - o the heauristic of n is less the cost of reaching n' from n plus the heuristic of n'
- time = exponential in worst case, but $O(bm)$ if heurictis is perfect
- space = exponential
- ** problem is that A* keeps most generated nodes in memory so eventually runs out of memory

IDA* SEARCH - Iterative deepening A*

- Works like IDS but uses f -cost instead of depth at each iteration
- Complete and optimal

SMA* - Simplified Memory-Bounded A*

- Works like A* but when runs out of memory drops worst f -valued leaf and expands to newest
- Optimal and complete if depth of shallowest goal node is $<$ memory size

MODULE 4 – CONSTRAINT SATISFACTION

A **constraint satisfaction problem (CSP)** is defined by $\{V, D, C\}$ where

- $V = \{V_1, \dots, V_n\}$ is a set of variables
- $D = \{D_1, \dots, D_n\}$ is the set of domains, D_i is the domain of possible values for variable V_i
- $C = \{C_1, \dots, C_m\}$ is the set of constraints

A **state** is an assignment of values to some or all the variables $\{V_i = x_i, V_j = x_j\}$

An assignment is **consistent** if it does not violate any constraints

A **solution** is a complete, consistent assignment (“hard constraints”)

- “soft constraints” are objective functions to be optimized

Examples: 8-Queens, Map Colouring, Street Puzzle, Scheduling, 3-SAT

Properties:

- Types of variables:
 - o Discrete and finite (map colouring, 8-queens, boolean CSP)
 - o Discrete variables with infinite domains (scheduling)
 - o Continuous domains (linear programming)
- Types of constraints:
 - o Unary – relates a single variable to a value
 - o Binary – relates two variables
 - o Higher order

If all domains have size d , then there are $O(d^n)$ complete assignments

Commutative – the order of any given set of actions has no effect on the outcome

BACKTRACKING SEARCH

```
function BACKTRACKING-SEARCH( csp ) returns a solution, or failure
  return RECURSIVE-BACKTRACKING( {}, csp )
```

```
function RECURSIVE-BACKTRACKING( assignment, csp ) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp )
  for each value in ORDER-DOMAIN-VALUES( var, assignment, csp ) do
    if value is consistent with assignment according to Constrains[csp] then
      add { var = value } to assignment
      result ← RECURSIVE-BACKTRACKING( assignment, csp )
      if result ≠ failure then return result
      remove { var = value } from assignment
  return failure
```

*Depth first search which chooses values for one variable at a time

FORWARD CHECKING

- Detecting failure early
- Keep track of remaining legal values (domain) for unassigned variables

** Most constrained variable, most constraining variable, least constraining variable

MODULE 5 – LOCAL SEARCH

Constructive methods: start from scratch and build up a solution

Iterative improvement methods: start with a solution and try to improve it

Hillclimbing:

1. Start at some initial configuration S
2. Let $V = \text{Eval}(S)$
3. Let $N = \text{MoveSet}(S)$ \rightarrow all the next possible moves
4. For each $X_i \in N$
 Let $V_{\max} = \max \text{Eval}(X_i)$ \rightarrow find the next max/improvement val
 And $X_{\max} = \text{argmax}_i \text{Eval}(X_i)$ \rightarrow find the next max/improvement pos
5. If $V_{\max} \leq V$, return S
6. Let $S = X_{\max}$ and $V = V_{\max}$. Go to 3

- Easy to program
- Requires no memory of where we have been
- Not complete because it never makes any downhill moves and can get stuck on a local max/min

Example: WALKSAT (flip one variable to see if it improves the satisfiability)

- Complete, but very inefficient

Simulated Annealing

1. Let S be the initial configuration and $V = \text{Eval}(S)$
2. Let i be a random variable move from moveset and let S_i be the next configuration $V_i = \text{Eval}(S_i)$
3. If $V < V_i$
 then $S = S_i$ and $V = V_i$
4. Else
 with probability p , $S = S_i$ and $V = V_i$
5. Go to 2 until you are bored

This allows for accepting a bad move with some low probability.

Boltzmann distribution: $T > 0$ is a parameter called temperature.

It starts high and decreases over time towards 0.

$\text{Exp}(-(V-V_i)/T)$

If T is decreased slowly enough, then simulated annealing is guaranteed (in theory) to reach best solution.

- When T is high: *Exploratory phase* (random walk)
- When T is low: *Exploitation phase* (randomized hill climbing)

GENETIC ALGORITHMS

- An encoded candidate solution is an **individual**
- Each individual has a **fitness** which is a numerical value associated with its quality of solution
- A **population** is a set of individuals
- Populations change over **generations** by applying operations to them

1. Initialize: population P consists of N random individuals (bit strings)
2. Evaluate: for each $x \in P$, compute $\text{fitness}(x)$
3. Loop:
 - a. For $i = 0$ to N do
 - Choose 2 parents, each with probability proportional to fitness scores
 - **Crossover** the 2 parents to produce a new bit string (child)
 - With some small probability **mutate** child
 - Add child to the population
4. Until some child is fit enough or you get bored
5. Return the best child in the population according to fitness function

Crossover:

- Cut the 2 individuals at a specific point, and swap pieces to create 2 potential children
- $(a \wedge m) \vee (b \wedge m)$ and $(a \wedge \sim m) \vee (b \wedge m)$ where m is a crossover mask

Mutation: allows us to generate desirable features that are not present in the original population

- allows to get out of local optima

Location search is useful for optimization problems

* Hill climbing always moves in the (locally) best direction

MODULE 6 – UNCERTAINTY

Reasons why logic fails:

- we are lazy (too much work to write down all antecedents and consequences)
- theoretical ignorance (there might not always be a complete theory)
- practical ignorance (uncertainty about particular instances – not enough information)

A **discrete random variable** describes an outcome that cannot be determined in advance

- these values may come from a countable domain

An **event** is a complete specification of the state of the world in which the agent is uncertain

- subset of the sample space
- must be mutually exclusive and exhaustive (at least one event must be true)

We let $\Pr(A)$, **probability** of A, denote the “degree of belief” we have that statement A is true

Probability distribution:

- A specification of a probability for each event in our sample space
- Probabilities must sum to 1

Joint probability distribution:

- Specification of probabilities for all combinations of events
- $\Pr(A = a \wedge B = b)$

Marginalisation (sumout rule):

- $\Pr(A = a) = \sum_b \Pr(A = a \wedge B = b)$

Conditional probability:

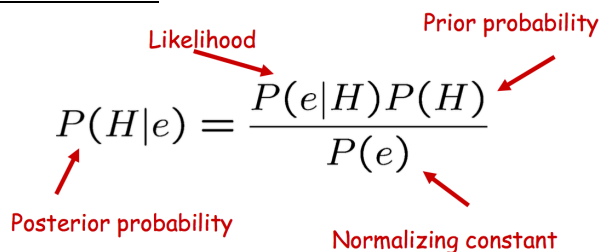
- fraction of worlds in which A is true given that B is also true
- $\Pr(A|B) = \frac{\Pr(A \wedge B)}{\Pr(B)} = \frac{\text{Probability of both events happening}}{\text{Probability of B happening by itself}}$

Inference : A conclusion reached on the basis of evidence and reasoning

** important note:

$$\Pr(A|B)\Pr(B) = \Pr(A \wedge B) = \Pr(B \wedge A) = \Pr(B|A)\Pr(A)$$

BAYES RULE


$$P(H|e) = \frac{P(e|H)P(H)}{P(e)}$$

Annotations: Posterior probability points to $P(H|e)$; Likelihood points to $P(e|H)$; Prior probability points to $P(H)$; Normalizing constant points to $P(e)$.

- Helps to form a hypothesis about the world based on observations

$$P(A = v_i|B) = \frac{P(B|A = v_i)P(A = v_i)}{\sum_{k=1}^n P(B|A = v_k)P(A = v_k)}$$

MODULE 7 – PROBABILISTIC REASONING

$\Pr(X)$ for variable X refers to the distribution over X

$\Pr(X | Y)$ refers to the family of conditional distributions over X , one for each $y \in \text{Dom}(Y)$

Probabilistic inference – given a prior distribution $\Pr(X)$ over variables X of interest, representing degrees of belief, and given a new evidence $E = e$, for some variable E , revise your degrees of belief (posterior) $\Pr(X | E = e)$

Two variables X and Y are **conditionally independent** given variable Z if we have:

$$\Pr(X = x | Z = z) = \Pr(X = x | Y = y, Z = z)$$

\Leftrightarrow

$$\Pr(X = x, Y = y | Z = z) = \Pr(X = x | Z = z) \Pr(Y = y | Z = z)$$

Inference example:



Computing $\Pr(g)$ in more concrete terms:

$$\Pr(c) = \Pr(c|e) \Pr(e) + \Pr(c|\sim e) \Pr(\sim e)$$

$$= 0.8 * 0.7 + 0.5 * 0.3 = 0.78$$

$$\Pr(\sim c) = \Pr(\sim c|e) \Pr(e) + \Pr(\sim c|\sim e) \Pr(\sim e) = 0.22$$

$$\Pr(\sim c) = 1 - \Pr(c), \text{ as well}$$

$$\Pr(g) = \Pr(g|c) \Pr(c) + \Pr(g|\sim c) \Pr(\sim c)$$

$$= 0.3 * 0.78 + 1.0 * 0.22 = 0.454$$

$$\Pr(\sim g) = 1 - \Pr(g) = 0.546$$

In this example, we can see that S is not independent of E , C , G or L .

However, if we learnt L , then S would be independent of E , C and G .

➔ So S is independent of E , C and G given L

So we have that

$$\Pr(S, L, G, C, E) = \Pr(S|L) \Pr(L|G) \Pr(G|C) \Pr(C|E) \Pr(E)$$

Our joint probability can be specified by local conditional distributions

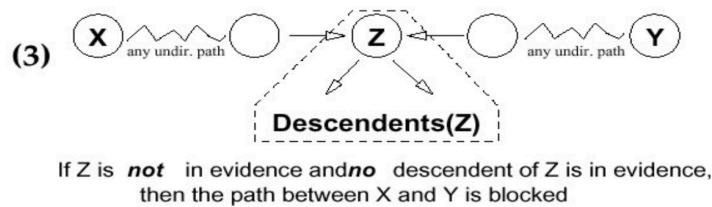
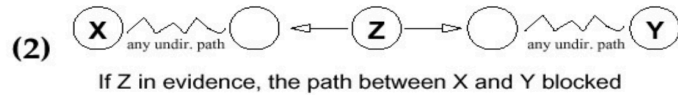
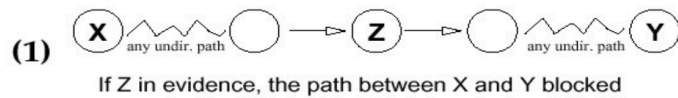
Bayesian Networks (BN)

- Exploits conditional independences for representation and inference
- Graphical representation of the **direct dependencies** over a set of variables + a set of **conditional probability tables (CPTs)** quantifying the strength of those influences
- Every X_i is conditionally independent of all of its non-descendants given its parents
- A BN over variables $\{X_1, X_2, \dots, X_n\}$ consists of:
 - o A DAG (direct acyclic graph) whose nodes are the variables
 - o A set of CPTs for each X_i

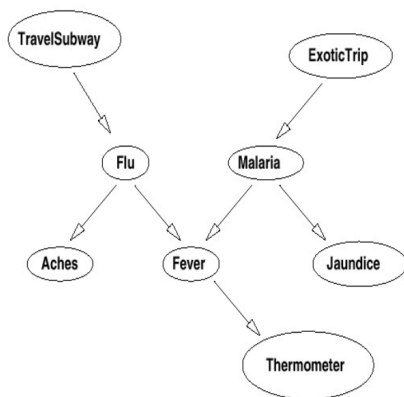
In a BN we have that every X_i is conditionally independent of all of its non-descendants given its parents

D-Separation: (to test independence) a set of variables E d-separates X and Y if it blocks every undirected path in the BN between X and Y

- X and Y are conditionally independent given evidence E if E d-separates X and Y



MODULE 8 – BAYESIAN NETWORKS



SIMPLE FORWARD INFERENCE:

When we have “upstream” evidence (from parents)

$$\begin{aligned}
 \Pr(J) &= \sum_{M,ET} \Pr(J, M, ET) \\
 &= \sum_M \Pr(J|M) \sum_{ET} \Pr(M|ET) \Pr(ET)
 \end{aligned}$$

$$\begin{aligned}
 \Pr(J|ET) &= \sum_M \Pr(J|M) \Pr(M|ET)
 \end{aligned}$$

Example:

$$\begin{aligned}
 P(\text{Fev}) &= \sum_{\text{Flu}, \text{M}, \text{TS}, \text{ET}} P(\text{Fev}, \text{Flu}, \text{M}, \text{TS}, \text{ET}) \\
 &= \sum_{\text{Flu}, \text{M}, \text{TS}, \text{ET}} P(\text{Fev}|\text{Flu}, \text{M}, \text{TS}, \text{ET}) P(\text{Flu}|\text{M}, \text{TS}, \text{ET}) \\
 &\quad P(\text{M}|\text{TS}, \text{ET}) P(\text{TS}|\text{ET}) P(\text{ET}) \\
 &= \sum_{\text{Flu}, \text{M}, \text{TS}, \text{ET}} P(\text{Fev}|\text{Flu}, \text{M}) P(\text{Flu}|\text{TS}) P(\text{M}|\text{ET}) P(\text{TS}) P(\text{ET}) \\
 &= \sum_{\text{Flu}, \text{M}} P(\text{Fev}|\text{Flu}, \text{M}) [\sum_{\text{TS}} P(\text{Flu}|\text{TS}) P(\text{TS})] \\
 &\quad [\sum_{\text{ET}} P(\text{M}|\text{ET}) P(\text{ET})]
 \end{aligned}$$

- (1) by marginalisation; (2) by the chain rule;
 (3) by conditional independence; (4) by distribution
 - note: all terms are CPTs in the Bayes net

$$P(\text{Fev}|\text{ts}, \sim\text{m}) = \sum_{\text{Flu}} P(\text{Fev}|\text{Flu}, \sim\text{m}) P(\text{Flu}|\text{ts})$$

SIMPLE BACKWARD INFERENCE

When we have “downstream” evidence (from children)

Here we are required to use Bayes rule.

Example:

$$\begin{aligned}
 P(\text{ET}|\text{j}, \text{fev}) &= \alpha P(\text{j}, \text{fev}|\text{ET}) P(\text{ET}) \\
 &= \alpha \sum_{\text{M}, \text{FI}, \text{TS}} P(\text{j}, \text{fev}, \text{M}, \text{FI}, \text{TS}|\text{ET}) P(\text{ET}) \\
 &= \alpha \sum_{\text{M}, \text{FI}, \text{TS}} P(\text{j}|\text{fev}, \text{M}, \text{FI}, \text{TS}, \text{ET}) P(\text{fev}|\text{M}, \text{FI}, \text{TS}, \text{ET}) \\
 &\quad P(\text{M}|\text{FI}, \text{TS}, \text{ET}) P(\text{FI}|\text{TS}, \text{ET}) P(\text{TS}|\text{ET}) P(\text{ET}) \\
 &= \alpha P(\text{ET}) \sum_{\text{M}} P(\text{j}|\text{M}) P(\text{M}|\text{ET}) \sum_{\text{FI}} P(\text{fev}|\text{M}, \text{FI}) \sum_{\text{TS}} \\
 &\quad P(\text{FI}|\text{TS}) P(\text{TS})
 \end{aligned}$$

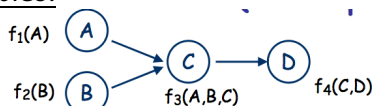
What is α ?

A **factor** is a CPT, a function $F(X_1, X_2, \dots, X_K)$

Variable Elimination

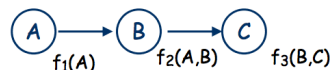
1. Restrict all factors if in evidence
2. Choose an elimination ordering
3. Sumout variables in provided order
 - a. For each variable, multiply all the factors that contain that variable to compute a new factor
 - b. Using that new factor, sumout the variable
4. The remaining factors are all in respect to query variable, so take their product and then normalize

VE Examples:



$$\begin{aligned}
 P(D) &= \sum_{A,B,C} P(D|C) P(C|B,A) P(B) P(A) \\
 &= \sum_C P(D|C) \sum_B P(B) \sum_A P(C|B,A) P(A) \\
 &= \sum_C f_4(C,D) \sum_B f_2(B) \sum_A f_3(A,B,C) f_1(A) \\
 &= \sum_C f_4(C,D) \sum_B f_2(B) f_5(B,C) \\
 &= \sum_C f_4(C,D) f_6(C) \\
 &= f_7(D)
 \end{aligned}$$

Define new factors: $f_5(B,C)$, $f_6(C)$, $f_7(D)$, in the obvious way



$$\begin{aligned}
 P(A|c) &= \alpha P(A) P(c|A) \\
 &= \alpha P(A) \sum_B P(c|B) P(B|A) \\
 &= \alpha f_1(A) \sum_B f_3(B,c) f_2(A,B) \\
 &= \alpha f_1(A) \sum_B f_4(B) f_2(A,B) \\
 &= \alpha f_1(A) f_5(A) \\
 &= \alpha f_6(A)
 \end{aligned}$$

New factors: $f_4(B) = f_3(B,c)$; $f_5(A) = \sum_B f_2(A,B) f_4(B)$; $f_6(A) = f_1(A) f_5(A)$

- Number of iterations is linear in number of variables
- Complexity is linear in number of variables and exponential in size of the largest factor

Relevance

1. A query is relevant
2. Parents of a relevant node are all relevant
3. If a node is in evidence and its parents are relevant, then it is also relevant

MODULE 9 – UTILITY THEORY

“Preferences over alternatives”

Preference Ordering

- $s \geq t$ s is at least as good as t
- $s > t$ s is strictly preferred to t
- $s \sim t$ agent
is indifferent between states s and t

When an agent's actions are deterministic, we know what states will occur

When an agent's actions are not deterministic, we represent preferences by **lotteries**

AXIOMS

1. Orderability

$$(A > B) \vee (B > A) \vee (A \sim B)$$

2. Transitivity

$$(A > B) \wedge (B > C) \Rightarrow (A > C)$$

3. Continuity

(if we strictly prefer A to B and B to C , then there is some probability for which we feel indifferent to getting A or B in comparison to C)

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

4. Substitutability

(If an agent is indifferent between A and B , then they are indifferent between more complex lotteries where A could be substituted with B)

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

5. Monotonicity

(If an agent prefers A to B , then they will prefer the lottery that has a high probability of A)

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

6. Decomposability

(reducing compound lotteries to simpler ones)

$$[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; q(1-p), B; (1-p)(1-q), C]$$

A **utility function** $U: S \rightarrow R$ associates a real valued **utility** with each outcome

- $U(s)$ measures your degree of preference for s
- U indices a preference ordering \geq_U over S defined as: ($s \geq_U t$) iff $U(s) \geq U(t)$

$Pr_d(s)$ is the probability of outcome s under decision d

The **expected utility** of decision d is defined as

$$EU(d) = \sum_{s \in S} Pr_d(s)U(s)$$

The sum of all utilities of every outcome under decision d

Example:

Supposed when a robot pours coffee, it spills 20% of the time, making a mess

Preferences: $(c, \sim \text{mess}) > (\sim c, \sim \text{mess}) > (c, \text{mess})$

If $U(c, \sim \text{ms}) = 10$, $U(\sim c, \sim \text{ms}) = 5$, $U(c, \text{ms}) = 0$,

then $EU(\text{getcoffee}) = (0.8)(10) + (0.2)(0) = 8$ and $EU(\text{donothing}) = 5$

A **decision problem under certainty** is:

- A set of **decisions** D
- A set of **outcomes** or states S
- An **outcome function** $\text{Pr}: D \rightarrow \Delta(S)$
 - o $\Delta(S)$ is the set of distributions over S (e.g. Pr_d)
- A **utility function** U over S

A **solution** to a decision problem is any $d^* \in D$ such that $EU(d^*) \geq EU(d)$ for all $d \in D$

The **principle of maximum utility (MEU)** states that the optimal decision under conditions of uncertainty is that with the greatest expected utility.

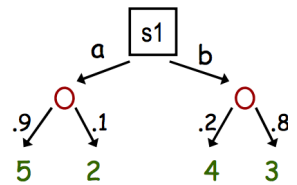
- U doesn't need to be unique -> it is unique up to a positive affine transformation
- Use Bayes Network because outcome space is large
- Use dynamic programming methods to construct optimal plans (**policies**) because our decision space is large (sequential choices)

We want to consider **policies** instead of sequence of actions

- If we have sequences of length k with action set A , we have A^k number of plans
- If we have n actions and m outcomes per action, then we have $(nm)^k$ policies

We use **Decision Trees**

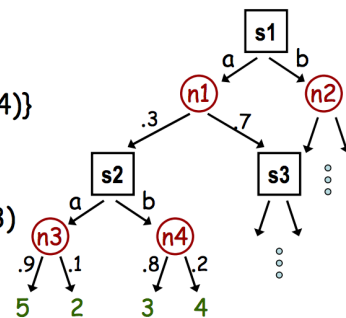
- Squares are decision nodes, action choices by decision maker
- Circles denote chance nodes (uncertainty – nature chooses child)
- Terminal nodes labeled with utilities



At any choice node, the decision maker chooses action that leads to **highest utility child**

Evaluation Decision Trees

- $U(n3) = .9 \cdot 5 + .1 \cdot 2$
- $U(n4) = .8 \cdot 3 + .2 \cdot 4$
- $U(s2) = \max\{U(n3), U(n4)\}$
 - decision a or b (whichever is max)
- $U(n1) = .3U(s2) + .7U(s3)$
- $U(s1) = \max\{U(n1), U(n2)\}$
 - decision: max of a, b



A **policy** assigns a decision to each choice node in tree

Two policies are **implementationally indistinguishable** if they disagree only at unreachable decision nodes

- Evaluate only $O((nm)^d)$ nodes in tree of depth d
- There are $(nm)^d$ policies so total computation for explicitly evaluating each policy would be $O(n^d m^{2d})$ using dynamic programming

****Computational Issues**

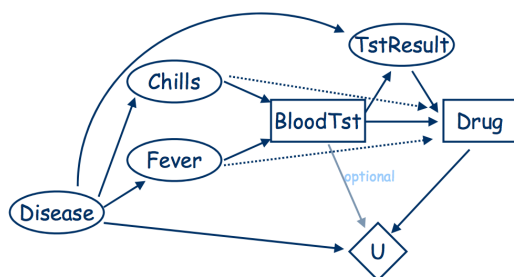
MODULE 10 – DECISION NETWORKS

Decision networks provide a way of representing sequential decision problems

- **Chance nodes** – random variables, probabilistic dependence on parents (circles)
- **Decision nodes** – variables set by decision maker, parents reflect **information available** at time decision is made (squares)
 - o Agent can make different decisions for each instantiation of parents (policies)
 - o These nodes are ordered
 - o **No-forgetting property** – any information available when a node is made is also available for its children (dashes)
- **Value nodes** – specifies utility of a state, depends only on state of parents of value node
 - o Only one value node in a decision network (diamond)

We have that a **policy** δ is a set of mappings δ_i , one for each decision node D_i

- $\delta_i : \text{Dom}(\text{Par}(D_i)) \rightarrow \text{Dom}(D_i)$
- so we have that δ_i associates a decision with each parent association for D_i



Ex: Policy for BT might be:

$$\begin{aligned}\delta_{BT}(c, f) &= bt \\ \delta_{BT}(c, \sim f) &= \sim bt \\ \delta_{BT}(\sim c, f) &= bt \\ \delta_{BT}(\sim c, \sim f) &= \sim bt\end{aligned}$$

Value of policy δ is the expected utility given that decisions are executed according to δ

- $EU(\delta) = \sum_x P(x, \delta(x)) U(x, \delta(x))$

An **optimal policy** is a policy δ^* such that $EU(\delta^*) \geq EU(\delta)$ for all δ

- can use dynamic programming and variable elimination to compute it

Computing Best Policy

- we work backwards, last to first
- given a node D that has no decision that follow it, we instantiate each of its parents
- we compute the expected utility of each decision for D for each parent instantiation
 - o compute the expected utility using variable elimination
 - o the policy will be the max decision for each parent instantiation
- once D has been optimized, it can now be treated as a random variable
 - o so now for each instantiation of its parents, we now know what value the decision should take
 - o treat it as a new CPT – for a given parent instantiation x , D gets $\delta(x)$ with probability 1 (all other decisions get probability 0)

*expected value of information

The value of information derives from the fact that with the information, one's course of action can be changed to suit the actual situation.

- How much you're willing to pay to get information

MODULE 11 – REASONING OVER TIME

Static probability distribution – assuming the state of things do not change over time

We have a series of time slices, each which contain a set of random variables, some observable and some not.

With the set of state and evidence variables for a given problem decided on, the next step is to specify how the world evolves (**transition model**) and how the evidence variables get their values (**sensor model**).

The transition model specifies the probability distribution over the latest state variables, given the previous values, that is, $\Pr(X_t | X_0 \dots X_{t-1})$.

A problem is that $X_0 \dots X_{t-1}$ is as t increases, so this is solved by the **Markov assumption** - the current state depends on only a finite fixed number of previous states.

There is also the problem of infinitely many values of t , which is solved by a **stationary process** – a process of change that is governed by laws that do not themselves change over time.

We see that the direction of dependence goes from the actual state of the world to the sensor values because the world causes the sensors to take on particular values (the rain causes the umbrella to appear).

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^t \mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1}) \mathbf{P}(\mathbf{E}_i | \mathbf{X}_i).$$

The three terms on the right-hand side are the *initial state model* $\mathbf{P}(\mathbf{X}_0)$, the *transition model* $\mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1})$, and the *sensor model* $\mathbf{P}(\mathbf{E}_i | \mathbf{X}_i)$

Stochastic Process

(stochastic: randomly selected)

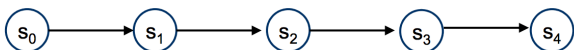
- Definition:
 - o Set of states S
 - o Stochastic dynamics $\Pr(s_t | s_{t-1}, \dots, s_0)$
 - o Can be viewed as a Bayes net with one random variable per time slice
- Problems:
 - o Infinitely many variables
 - o Infinitely large conditional probability tables
- Solutions:
 - o Stationary process: dynamics do not change over time
 - o Markov assumption: current state depends only on a finite history of past states

K-order Markov Process

- Assumption: last k states sufficient

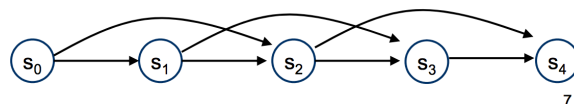
- **First-order Markov Process**

- $\Pr(s_t | s_{t-1}, \dots, s_0) = \Pr(s_t | s_{t-1})$



- **Second-order Markov Process**

- $\Pr(s_t | s_{t-1}, \dots, s_0) = \Pr(s_t | s_{t-1}, s_{t-2})$

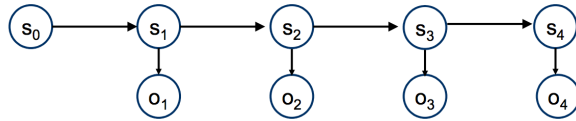


7

- Advantage: can specify entire process with finitely many time slices
- Two slices sufficient for a first-order Markov process

Hidden Markov Models

- Set of states S
- Set of observations O
- Transition model: $\Pr(s_t | s_{t-1})$
- Observation model: $\Pr(o_t | s_t)$
- Prior: $\Pr(s_0)$



Inference in Temporal Models

1. Monitoring

- $\Pr(s_t | o_t, \dots, o_1)$: distribution over current state given observations
- Example: robot localisation
- Forward algorithm : corresponds to variable elimination
 - Factors: $\Pr(s_0), \Pr(s_i | s_{i-1}), \Pr(o_i | s_i), 1 \leq i \leq t$
 - Restrict o_1, \dots, o_t to the observations made
 - Summout s_0, \dots, s_{t-1}
 - $\sum_{s_0 \dots s_{t-1}} \Pr(s_0) \prod_{1 \leq i \leq t} \Pr(s_i | s_{i-1}) \Pr(o_i | s_i)$

2. Prediction

- $\Pr(s_{t+k} | o_t, \dots, o_1)$: distribution over future state given observations
- Example: stock market prediction
- Forward algorithm : corresponds to variable elimination
 - Factors: $\Pr(s_0), \Pr(s_i | s_{i-1}), \Pr(o_i | s_i), 1 \leq i \leq t+k$
 - Restrict o_1, \dots, o_t to the observations made
 - Summout $s_0, \dots, s_{t+k-1}, o_{t+1}, \dots, o_{t+k}$
 - $\sum_{s_0 \dots s_{t+k-1}, o_{t+1} \dots o_{t+k}} \Pr(s_0) \prod_{1 \leq i \leq t+k} \Pr(s_i | s_{i-1}) \Pr(o_i | s_i)$

3. Hindsight

- $\Pr(s_k | o_t, \dots, o_1)$ for $k < t$: distribution over a past state given observations
- Example: crime scene investigation
- Forward algorithm : corresponds to variable elimination
 - Factors: $\Pr(s_0), \Pr(s_i | s_{i-1}), \Pr(o_i | s_i), 1 \leq i \leq t$
 - Restrict o_1, \dots, o_t to the observations made
 - Summout $s_0, \dots, s_{k-1}, s_{k+1}, \dots, s_t$
 - $\sum_{s_0 \dots s_{k-1}, s_{k+1} \dots s_t} \Pr(s_0) \prod_{1 \leq i \leq t} \Pr(s_i | s_{i-1}) \Pr(o_i | s_i)$

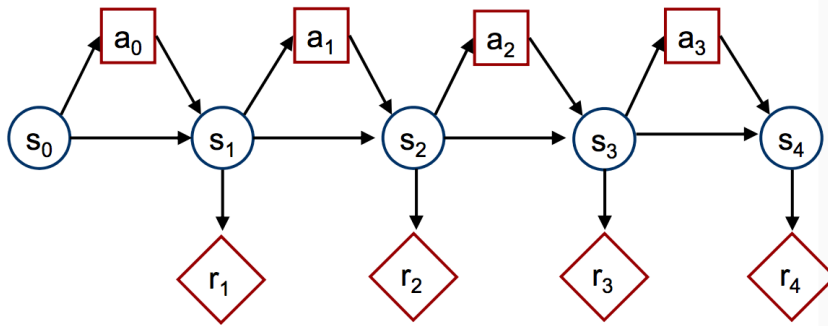
4. Most likely explanation

- $\text{Argmax}_{s_0 \dots s_t} \Pr(s_0, \dots, s_t | o_t, \dots, o_1)$: most likely state sequence given observations
- Example: speech recognition
- Viterbi algorithm: corresponds to a variant of variable elimination
 - Factors: $\Pr(s_0), \Pr(s_i | s_{i-1}), \Pr(o_i | s_i), 1 \leq i \leq t$
 - Restrict o_1, \dots, o_t to the observations made
 - Maxout s_0, \dots, s_t
 - $\max_{s_0 \dots s_t} \Pr(s_0) \prod_{1 \leq i \leq t} \Pr(s_i | s_{i-1}) \Pr(o_i | s_i)$

Dynamic Bayesian Network

- Encode states and observations with several random variables

MODULE 12 – MARKOV DECISION PROCESSES



Markov Decision Process

- Set of states: S
- Set of actions (decisions): A
- Transition model: $\Pr(s_t \mid a_{t-1}, s_{t-1})$
- Reward model (utility): $R(s_t)$
- Discount factor: $0 \leq \gamma \leq 1$
- Horizon (# of time steps): h

Goal: find optimal policy

Example: Inventory Management

- Markov Decision Process
 - States: *inventory levels*
 - Actions: *{doNothing, orderWidgets}*
 - Transition model: *stochastic demand*
 - Reward model: *Sales - Costs - Storage*
 - Discount factor: *0.999*
 - Horizon: *∞*
- Tradeoff: *increasing supplies decreases odds of missed sales but increases storage costs*

policy: choice of action at each time step

Policy evaluation:

$$EU(\delta) = \sum_{t=0}^{\infty} \gamma^t \Pr(s_t \mid \delta) R(s_t)$$

Optimal policy:

(the policy with the highest expected utility)

$$EU(\delta) \leq EU(\delta^*) \text{ for all } \delta$$

There are 3 algorithms to optimize policy:

1. Value iteration
 - This is equivalent to variable elimination
 - Performs dynamic programming
 - Optimize decisions in reverse order
2. Policy iteration
3. Linear programming