- agent function maps from precept history to actions $f: P^* \to A$
- agent program I runs on some machine M to implement f: $f = Agent(I, M)$

not every function can be implemented by some program

environment	agent requires
partially observ.	memory
stochastic	prepare for contingencies
multi-agent	behave randomly
static	has time to compute
continuous time	continuously w/ controller

Search problem consists of:
- state space
- set of actions for each state
- transition model
- step cost function
- start state + goal test

DFS
b = branching factor
m = depth
time: $O(b^m)$
space $O(bm)$
- not optimal

BFS
time: $O(b^s)$
space: $O(b^s)$
optimal if costs = 1

UCS - optimal
- solution cost = C^*
- arcs cost $\geq \epsilon$
$O(b^{C^*/\epsilon})$ time
$O(b^{C^*/\epsilon})$ space

Tree search - exponential
graph search - quadratic
- required mem prop to its runtime

Max of admissible heuristics is admissible + better
- admissible: heuristic(x) $\leq$ actual cost(x)
- consistent: $h(A) + h(C) \leq cost(A \text{ to } C)$
- $h(A) \leq cost(A \text{ to } C) + h(C)$

Hill climbing - just climb up

Local search - path is irrelevant - just find the solution
- improve your current state
- constant space, works for online + offline search

- nondeterminism
- solutions are contingency plans
- use AND-OR search to find them

outcomes don't depend deterministically on some hidden state

Local beam search
- k copies of local search w/ rand. starts
- for each iteration
  - generate all successors for k states
  - choose best k to be new states
- like evolution
- we can see here the grass is greener

Simulated annealing
- allow "bad" move depending on temp
- high temp = more bad moves allowed, gets out of local min
- gradually reduce temp
- finds global optimum w/ prob 1 if slow enough cooling sched

Sensorless blank state
- everything work the same

Minimax
- efficiency just like DFS
- we can't go all the way down so we have eval funcs

Alpha-beta pruning
- we're computing min-value at node n
- α is best value max can get
- if n becomes worse than α we can prune n's other children

doubles solvable depth

Partial observability
- belief state - set of all environments the agent could be in
- n physical states: 2^n belief states
- Transition model
  - determ: Result(b,a) = union of Result(s,a) for each s in b
  - nondetermin: Result(b,a) = union of results(s,a) for each s in b
- steplost(b, a, b') = steplost(s, a, s') for any s in b

CSPS
- constraint graphs
  - each constraint relates 2 variables
- types of CSPS
  - discrete variables
    - n variables, domain size d: $O(d^n)$ complete assignments
  - continuous variables
    - LP

types of constraints
- unary constraints: one variable
- binary constraints: two variables

Backtracking search
- one variable at a time
- check constraints as you go
DFS + these 2 improvements

Variable ordering for backtracking
- use minimum remaining value
- variable w/ fewest values left in domain
then do LCV (least constraining value)

Arc consistency - arc $X \to Y$ is consistent iff for every x in tail there is some y in head which could legally be assigned

$a \models b$ iff in every world where a is true, b is also true

Speedups for backtracking: ordering, filtering, structure

Successor state axiom: $X_t \Leftrightarrow [X_{t-1} \wedge \neg(\text{Some Action}_{t-1} \text{ made it false})] \vee [\neg X_{t-1} \wedge (\text{Some Action}_{t-1} \text{ made it true})]$

$P(a|b) = \frac{P(a, b)}{P(b)}$ $\quad P(y)P(x|y) = P(x, y)$ $\quad$ chain rule: $P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$

Bayes: $P(x|y) = \frac{P(y|x)}{P(y)} P(x)$

Independence: X and Y are ind. iff $\forall x,y$ $P(x,y) = P(x)P(y)$
$\forall x,y$ $P(x|y) = P(x)$ or $P(y|x) = P(y)$

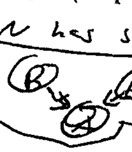
X is ind of Y given Z iff $\forall x,y,z$ $P(x|y,z) = P(x|z)$ $\quad \forall x,y,z$ $P(x,y|z) = P(x|z)P(y|z)$

CPT = conditional prob table

Bayes nets: n variables, max domain size $= d$, max parents $= k$
full joint distribution has size $O(d^n)$ | BN has size $O(n \cdot d^k)$
$P(X_1 \ldots X_n) = \prod_i P(X_i | parents(X_i))$

Variable elimination:
move summations inward
- start w/ initial factors
  while still hidden variables
  - pick hidden var H
  - join all factors mentioning H
  - eliminate (sum out) H
- join all remaining factors + normalize

time is linear in network size if you eliminate from leaves to root in a poly tree



B is ind of E
B not ind of E given A
J ind M given A

Markov models
$P(X_t | X_{t-1})$ tells you how the state evolves over time

Forward algorithm for markov models: $P(X_0 \ldots X_t) = P(X_0) \prod_t (P(X_t | X_{t-1}))$ - joint distribution
$P(X_t) = \sum_{x_{t-1}} P(X_{t-1} = x_{t-1}) P(X_t | X_{t-1} = x_{t-1})$

Stationary distribution:
$P_\infty = P_{\infty+1} = T^T P_\infty$

example:

X_{t-1}	$P(X_t \mid X_{t-1})$ sun	rain
sun	.9	.1
rain	.3	.7

solving for P_∞

$$\begin{bmatrix} .9 & .3 \\ .1 & .7 \end{bmatrix} \begin{bmatrix} p \\ 1-p \end{bmatrix} = \begin{bmatrix} p \\ 1-p \end{bmatrix}$$

$.9p + .3(1-p) = p$

HMMs: $X_0 \to X_1 \to X_2 \to X_3$, $X_1 \to E_1$, $X_2 \to E_2$, $X_3 \to E_3$
- initial dist: $P(X_0)$
- initial model: $P(X_t | X_{t-1})$
- sensor model: $P(E_t | X_t)$

joint dist: $P(X_0, X_1 \ldots X_T, E_T) = P(X_0) \prod_{t=1:T} P(X_t | X_{t-1}) P(E_t | X_t)$

more on back

more HMMs:

Filtering: $P(X_t \mid e_{1:t})$ – input to decision process of rational agent

prediction: $P(X_{t+k} \mid e_{1:t})$ $k>0$ – evaluation of possible action sequences

Smoothing: $P(X_k \mid e_{1:t})$ $0 \le k < t$ – better estimate of past states

most likely explanation: $\arg\max_{X_{1:t}} P(X_{1:t} \mid e_{1:t})$

filtering alg: $P(X_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid X_{t+1}) \sum_t P(x_t \mid e_{1:t}) P(X_{t+1} \mid x_t)$

DBN – repeat fixed Bayes net structure at each time
- every HMM is a single variable DBN
- every discrete DBN is an HMM
- DBNs have way fewer parameters than HMMs

Viterbi alg (max)
- for each state at t, keep track of max prob of any path to it

$P(e_{t+1} \mid X_{t+1}) \max_{x_t} P(X_{t+1} \mid X_t) m_{1:t}$

$O(|X|^2 T)$ time | $O(|X| T)$ space | $O(|X|^T)$ paths

$O(|\text{states}|^L)$ for tracking

- Decision network = Bayes nets + action nodes (can not have parents, given) + utility nodes (depends on action + chance nodes)

$EU = \sum_w P(w \mid e, a) U(a, w)$ for all a
- return action w/ highest EU

Value of Information
- change in expected utility by looking at variable
- non-negative
- non additive $VPI(E_i, E_j \mid e) \ne VPI(E_i \mid e) + VPI(E_j \mid e)$
- order independent $VPI(E_i, E_j \mid e) = VPI(E_j, E_i \mid e)$

$\text{Utility} = \sum_i P_i U(S_i)$

$MEU = \max \text{ utility}$

MDPs are fully observable but probabilistic search probs

$V^{\pi}(s) = P(S_1 \mid S_0, \pi(S_0)) \times P(S_2 \mid S_1, \pi(S_1))$ probabilities ← reward sequence for states + policy

- expected utility of π in $S_0 = V^{\pi}(S_0)$
  - sum over all possible state sequences of (discounted sum of rewards)(prob of state sequence)

- Neural networks
- inputs a_i come to neuron j from neuron i
- each has a weight $w_{i,j}$
- total input $in_j = \sum_i w_{i,j} a_i$
- output $a_j = g(in_j) = g(w \cdot a)$ g = activation function

($\cdot$ = dot product)

$Q^*(s,a)$ = expected utility of taking action a in state s and then acting optimally

$V^*(s) = \max_a Q^*(s,a)$ = expected utility of starting in s and acting optimally

So... $V^*(s) = \max_a \sum_{s'} P(s' \mid a, s') [R(s,a,s') + \gamma V^*(s')]$
- Bellman equation

Value Iteration

Do Bellman update until you converge (all updates smaller than $\epsilon(1-\gamma)/\gamma$)

$V_{k+1}(s) = \max_a \sum_{s'} P(s' \mid a, s) [R(s,a,s') + \gamma V_k(s')]$

ie $V_{k+1} = B V_k$

we know it will converge
- error is reduced by a factor of at least γ every iteration
  - exponentially fast

- for policy iteration
  - do VI
  - but keep track of best policy

- Perceptron learning (neural networks)
- if $y \ne h_w(x)$ (there is an error)
- if $w \cdot x < 0$ but output should be $y=1$
  - false neg
  - increase weights on pos inputs, decrease on neg inputs
- if $w \cdot x > 0$ but output should be $y=0$
  - false pos
  - decrease weights on pos, increase weights on neg
- $w = w + \alpha(y - h_w(x)) \times$ $\quad \alpha$ = learning rate
- data is lin sep if you can draw a line
  - if it is – perceptron learning will eventually converge to perfect separation
  - if not perceptron learning will converge to minimal error solution as long as α is decreased correctly

MLP = multi layer perceptron
- feed forward net w/ at least one hidden layer
- can rep any function

MLP's minimize squared error loss function

Bayesian learning
- predictions use likelihood weighted average over hypotheses

$P(X_{n+1} \mid x) = \sum_k P(X_{n+1} \mid x, h_k) P(h_k \mid x) = \sum_k P(X_{n+1} \mid h_k) P(h_k \mid x)$

We can express V and Q as weighted linear functions

$V_w(s) = w_1 f_1(s) + \ldots + w_n f_n(s)$ (approx)

~~$Q(s,a)$~~ ~~approx~~ $Q_w(s,a) = w_1 f_1(s,a) + \ldots + w_n f_n(s,a)$

updating: $w_i = w_i + \alpha [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] f_i(s,a)$

- this approximation may diverge!!

Q learning

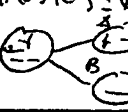
Sample $= R(s,a,s') + \gamma \max_{a'} Q(s',a')$

$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha)[\text{sample}]$
- converges to optimal policy even if you are acting suboptimally

Decision trees | choose greatest IG
- 2^{2^n} trees w/ n boolean outputs
- information gain H = entropy = # of bits you need to tell each piece of information

$H(\text{before}) - H(\text{after})$

$H(\text{after}) = (\text{percent in } A)(H(A)) + (\text{percent in } B)(H(B))$



$H(\langle P_1 \ldots P_n \rangle) = \sum_i -P_i \log_2 P_i$

$IG = H(\text{before}) = 1 \quad H(\text{after}) = \frac{3}{4}\left(-\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3}\right) + \frac{1}{4}(-1(\log(1)))$