Agents

- *Agent*: perceives environment through sensors and acts upon them through actuators
- PEAP
  - *Percepts*: observations of environment (made by sensors)
    - *Percept Sequence*: entire history of everything agent has perceived
  - *Environment*: where agent exists
  - *Action*: affects the environment
  - *Performance Measure*: desired environment state(s)
- *Rational* agent: Maximises Performance Measure based on percept sequence and built-in knowledge
- *Agent Function*: maps percepts to actions

Environment Properties (ODESD)

- Fully/Partially Observable
  - can sensors capture complete state of environment at every point in time
- Deterministic/Stochastic
  - does current state uniquely determine the next
- Episodic/Sequential
  - do short-term actions have long-term consequences
- Static/Dynamic
  - does environment change when agent does nothing
- Discrete/Continuous
  - finite/infinite number of possible actions
- Single/Multi Agent
  - are there other agents

Agent Architectures

Simple Reflex Agent	Model-based Reflex Agent
only chooses action based on current percept, ignores all preceding information	maintains internal state of environment - good for partially observable
Goal-Based Agent	Utility-Based Agent
makes decisions in order to achieve set of predefined goals, in addition to maintaining internal state	compares desirability of different environment states via utility function

Single-State Problems

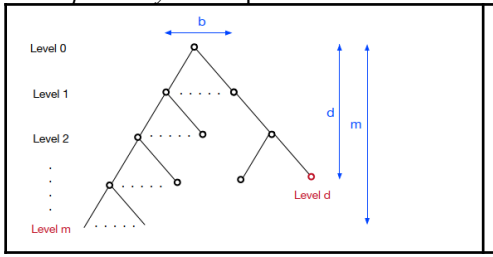
- *Single-State*: completely observable; certain about our current state
- *Problem*:
  1. Initial State
  2. Actions/Operators/Successor Function
  3. Goal Test
  4. Path Cost (additive)
- *Solution:* sequence of actions leading from initial state to goal state
- *State Space*: all possible states in domain; must be abstracted

Search Algorithms/Strategy

- offline, simulated exploration of state space by generating successors of already-explored states
- *Expand Function*: picks order of node expansion

Evaluation

- *Completeness*: guaranteed to find (any) solution?
- *Time Complexity*
- *Space Complexity*
- *Optimality*: find optimal solution?



- b: maximum branching factor
- d: depth of least cost solution
- m: maximum depth

- *State*: representation of physical configuration of environment state
- *Node*: data structure constituting part of search tree (parent, child, depth, path cost)

Uninformed Search Algorithms

- *Uninformed*: have access only to problem definition

Breadth-First Search	Uniform-Cost Search
expand shallowest unexpanded node	expand least-cost unexpanded node
Depth-First Search	Depth-Limited Search
expand deepest unexpanded node	DFS with depth limit
Iterative Deepening	Bidirectional Search
DLS with increasing depth limits combination of BFS and DFS	search (usually BST) simultaneously forwards from start point and backwards from goal

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Informed Search Algorithms

- *Informed*: have access to a heuristic function that estimates cost of solution from node
- *Heuristic Function*, h(n): decreases branching factor
  - relax problem definition, store precomputed solution costs for subproblems, learn from experience with problem classification

Best-First Search	Greedy Best-First Search
selects node according to evaluation function	expand node with minimal h(n)
A* Search	Recursive Best-First Search (RBFS)
- f(n) = cost to reach node + h(n) - *Admissible Heuristic:* never overestimates estimated cost > true	uses limited amounts of memory; can solve problems A* cannot solve because it runs out of memory; recursive stack

-

Criterion	Best-First	Greedy Best-First	A*	RBFS
Complete?	No	No	Yes	No
Time	$O(b^m)$	$O(b^m)$	$O(b^d)$	$O(b^m)$
Space	$O(b^m)$	$O(b^m)$	$O(b^d)$	$O(bm)$
Optimal?	No	No	Yes	No

- *Hill Climbing*:
  - finds for local minimum/maximum

Adversarial Search

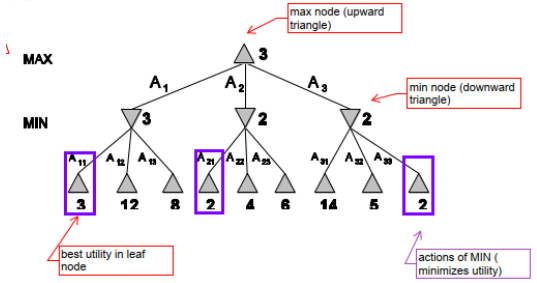
- *Problem*:
  1. Initial State
  2. Actions
  3. Terminal Test (win/lose/draw)
  4. Utility Function

Minimax

- **Complete**: Yes, if the tree is finite
- **Optimal**: Yes, against an optimal opponent
- **Time Complexity**: $O(b^m)$
  - expensive
- **Space Complexity**: $O(bm)$ (depth-first exploration)

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

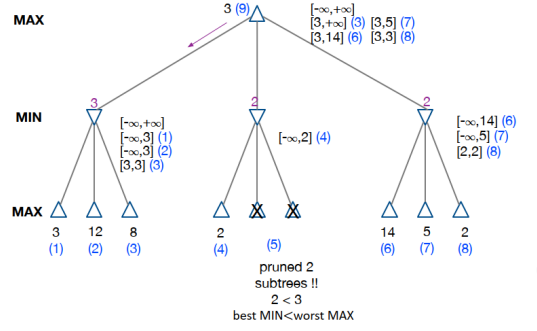
y game:




```
function MINIMAX-DECISION(game) returns an operator
    for each op in OPERATORS[game] do
        VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
    end
    return the op with the highest VALUE[op]

function MINIMAX-VALUE(state, game) returns a utility value
    if TERMINAL-TEST[game](state) then
        return UTILITY[game](state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Alpha-Beta Pruning



- Time Complexity
  - "perfect ordering": $O(b^{\frac{m}{2}})$

ExpectiMinimax

handles chance nodes

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

Learning Evaluation Function Weights

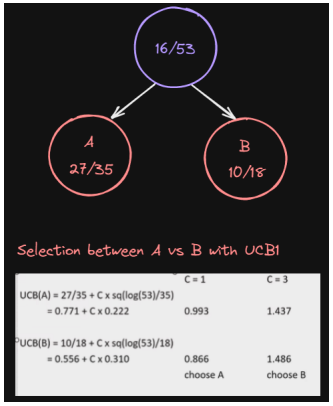
- Evaluation function typically use a *linear weighted sum of features*
  - $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$
  - $= \sum_{i=1}^{n} w_i f_i(s)$
  - $= w \cdot f(s)$

- *Supervised Learning*:
  - CONS: delayed reinforcement; credit assignment (which action responsible for win/loss?); only takes current state into consideration
- *Temporal Difference Learning*: reinforcement learning; multi-step prediction; correctness of prediction not known until several steps later
- TDLeaf(λ): uses temporal difference learning with Minimax

MCTS

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

- $UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(Parent(n))}{N(n)}}$
  - ExploitationTerm: $\frac{U(n)}{N(n)}$
  - Exploration term: $\sqrt{\frac{\log N(Parent(n))}{N(n)}}$
  - $U(n)$: total utility (eg. no of wins) from all playouts through node n
  - $N(n)$: total number of playouts through node n
  - $Parent(n)$: parent node of n in the tree
  - C a constant that balances exploration term and derivation term
    - high = more adventurous and prefers exploration
  - Choose node with highest $UCB1$ value

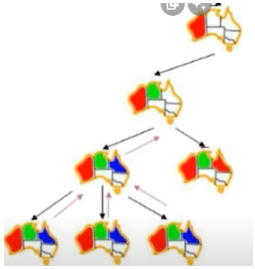
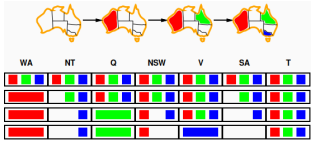


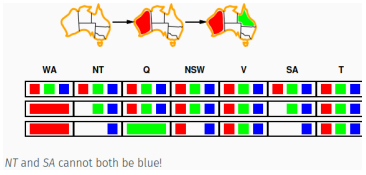
1. Expansion: create new node
2. Simulation: playout to terminal state
3. Backpropagation: update statistics

Constraint Satisfaction Problems

- general-purpose algorithms with more power than standard search

- *state*
  - set of **variables** $X = \{X_1, ..., X_n\}$
  - set of **domains** for each variable X, $D = \{D_1, ..., D_n\}$
- *goal test*
  - set of **constraints**
    - restrictions on domain of each variables

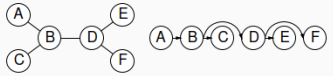
- *Backtracking Search*: DFS with single-variable assignment
  - 
  -
  -
- *Minimum Remaining Values (MRV)*: choose variable with fewest legal values
- *Degree Heuristic*: choose variable with most constraints on remaining values
- *Least Constraining Value*: rules out fewest values in remaining values
- *Forward Checking*: keep track of remaining legal values for unassigned variables; terminate when any variable has no legal values
  - 

  -
- *Constraint Propagation*: repeatedly enforce constraints locally – detects failures earlier (checks each variable's arc consistency with every other variable)
  - 

  - *NT and SA cannot both be blue!*
- *Arc Consistency*: every value for A, there is value for B such that both satisfy constraint
  - $X \rightarrow Y$ is arc consistent iff
    for every value x of X there is at least one value y of Y
    that satisfies the constraint between X and Y
- *Tree-structured CSPs*

  Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$
  - worst case CSP: $O(d^n)$

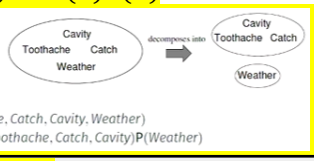
1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



2. For j from n down to 2, apply MAKEARCCONSISTENT($Parent(X_j), X_j$)
3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

-

Bayesian Networks

Chain Rule	Independence
$P(X_1, \ldots, X_n) = P(X_1, \ldots, X_{n-1})\ P(X_n \mid X_1, \ldots, X_{n-1})$ $= P(X_1, \ldots, X_{n-2})\ P(X_{n-1} \mid X_1, \ldots, X_{n-2})\ P(X_n \mid X_1, \ldots, X_{n-1})$ $= \ldots$ $= \prod_{i=1}^{n} P(X_i \mid X_1, \ldots, X_{i-1})$	A and B are independent iff $P(A\|B) = P(A)$ or $P(B\|A) = P(B)$ $P(A, B) = P(A)P(B)$  P(Toothache, Catch, Cavity, Weather) = P(Toothache, Catch, Cavity)P(Weather)
Conditional Probability	**Bayes' Rule**
$P(a\|b) = \frac{P(a \wedge b)}{P(b)}\ if\ P(b) \neq 0$	$P(a\|b) = \frac{P(b\|a)P(a)}{P(b)}$

- *Bayesian Network*: set of nodes (one per variable); direct, acyclic; conditional distribution for each node given its parents, $P(X_i|Parents(X_i))$
  - simplest case: conditional probability table (CPT) giving distribution over X_i for each combination of parent values

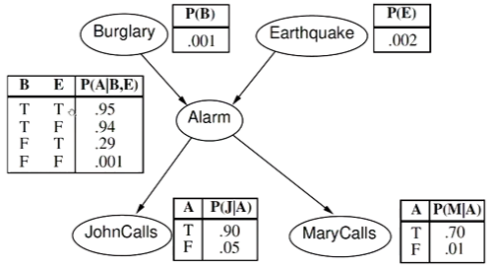


table has 2^k rows, k=num parents
network requires $O(n \cdot 2^k)$ vs $O(2^n)$ for joint distribution
- 1+1+4+2+2=**10** vs 2^5-1=**31**

Full Joint Distribution:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(x_i|\text{parents}(X_i))$$

Construction
1. Choose an ordering of variables X_1, ..., X_n
2. For $i = 1$ to n
   - add X_1 to the network
   - select parents from X_1, ..., X_n such that $P(X_1|Parents(X_1)) = P(X_1|X_1,\ldots X_{i-1})$
- Choice of parents guarantees the global semantics:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | X_1, \ldots, X_{i-1}) \quad \text{(chain rule)}$$
$$= \prod_{i=1}^{n} P(X_i | \text{Parents}(X_i)) \quad \text{(by construction)}$$

- Order: causes→symptoms

Hidden Markov Model

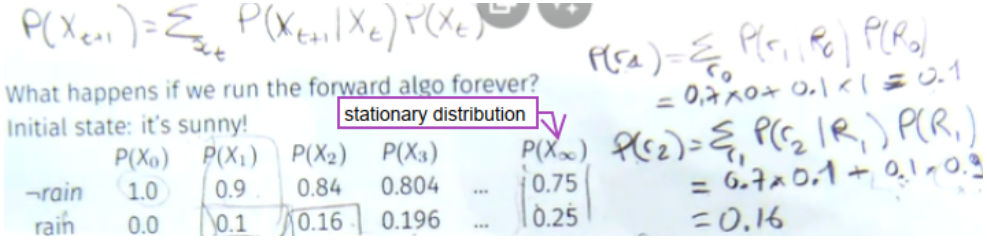
Markov Models

- *Markov Assumption*: current state depends only on finite fixes number of states
- *First-order:* only considers previous state
  - $P(X_t | X_{0:t-1}) = P(X_t | X_{t-1})$
- *higher order*: still same probability distribution at each time step

$\Rightarrow P(x_{t+1}) = \sum_{x_t} P(x_{t+1} | X_t) P(X_t)$

Stationary Distribution

$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x) P_\infty(x)$



Hidden Markov Models

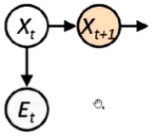
Joint Distribution

$$P(X_{1:t}, E_{1:t}) = P(X_1) P(E_1 | X_1) \prod_{i=2}^{t} P(X_i | X_{i-1}) P(E_i | X_i)$$

- State and Evidence independent of all past states and evidence, given the current state
- *Filtering/State Estimation*: with evidence gathered, can I estimate current state?
- *Prediction*: Given evidence gathered, can I determine future state?
- *Smoothing*: Given evidence up till now, can I compute a past state?
- *Most likely explanation*: given evidence gathered up till now, what are the most likely sequence of states that can explain my current state?
- *Learning*: given evidences, can we update the transition and sensor models

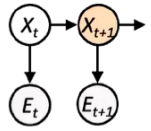
Forward Algorithm

- Prediction
  - $P(X_{t+1} | e_{1:t}) = \sum_{x_t} \underbrace{P(X_{t+1} | x_t)}_{\text{transition model}} \underbrace{P(x_t | e_{1:t})}_{\text{recursion}}$
- Filerting
  - $= \alpha \underbrace{P(e_{t+1} | X_{t+1})}_{\text{sensor model}} \sum_{x_t} \underbrace{P(X_{t+1} | x_t)}_{\text{transition model}} \underbrace{P(x_t, e_{1:t})}_{\text{reccursion}}$
  - $P(X_{t+1} | e_{1:t}) = \sum_{x_t} P(X_{t+1}, x_t | e_{1:t})$
    $= \sum_{x_t} P(X_{t+1} | x_t, e_{1:t}) P(x_t | e_{1:t})$
    $= \sum_{x_t} \underbrace{P(X_{t+1} | x_t)}_{\text{transition model}} \underbrace{P(x_t | e_{1:t})}_{\text{recursion}}$



Filtering

$$P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1})$$
$$= \alpha P(e_{t+1} | X_{t+1}, e_{1:t}) P(X_{t+1} | e_{1:t})$$
$$= \alpha \underbrace{P(e_{t+1} | X_{t+1})}_{\text{update}} \underbrace{P(X_{t+1} | e_{1:t})}_{\text{prediction}}$$



$$P(X_{t+1} | e_{1:t+1}) = \alpha P(e_{t+1} | X_{t+1}) P(X_{t+1} | e_{1:t})$$
$$= \alpha P(e_{t+1} | X_{t+1}) \sum_{x_t} P(X_{t+1} | x_t, e_{1:t}) P(x_t, e_{1:t})$$
$$= \alpha \underbrace{P(e_{t+1} | X_{t+1})}_{\text{sensor model}} \sum_{x_t} \underbrace{P(X_{t+1} | x_t)}_{\text{transition model}} \underbrace{P(x_t, e_{1:t})}_{\text{reccursion}}$$

- α is a normalizing constant: make probabilities sum to 1

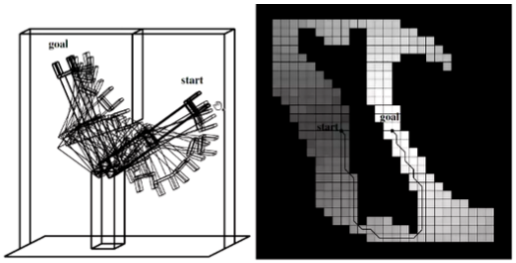
Robotics

Uncertainty

- *Localisation*
  - *Sensor Model*: Use observations of landmark to estimate state of robot
  - *Motion Model*: Update state using its movements
- *Mapping*

Motion Planning

- *Cell Decomposition:* divide space into simple cells; each can traverse "easily" (eg. convex); discretise configuration space into cells; do search from start to end
  - CONS: does not capture details in config space



Problem: may be no path in pure freespace cells
Solution: recursive decomposition of mixed (free+obstacle) cells

- *Skeletonization*: finite number of easily connected points/lines; form a graph such that any two points are connected by a path on the graph
  - Voronoi Diagrams: locus of points equidistant from obstacles
  - CON: does not scale well to higher dimensions
  - 
- *Probabilistic Roadmap*: generate random points in C-space, keeping those in freespace; create graph by joining pairs by straight lines
  - 
  - CON: generating enough points to ensure that every start/goal pair is connected through the graph