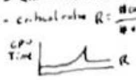
Matthew Tran
CS188
MT1

- Reflex Agent - pick action based on current state
- Planning Agent - use model & simulate actions
- Search Problem
  - State Space - all poss. states
  - Successor Function - input: state, action; output: cost, successor state
  - start state
  - goal test
- World state - all info about world
- Search state - only info needed for problem
- state space graph - each possible state / node, directed
- search tree - state can appear > 1 time
- Uninformed Search
  - idea: fringe, expand, goal test
  - completeness - if solution, find it?
  - optimality - lowest cost?
  - branching factor b - ↑ nodes w/ each expand, $O(b^k)$ at depth k ↗
  - maximum depth m
  - depth of shallowest solution s
  - Depth First Search (DFS)
    - idea: select deepest node to expand
    - implementation: LIFO stack
    - complete? NO   Note: all worst case
    - optimal? NO
    - time? $O(b^m)$
    - space? $O(bm)$ // [illegible]
  - Breadth First Search (BFS)
    - idea: shallowest node expand
    - implementation: FIFO queue
    - complete? YES
    - optimal? NO, unless costs equal
    - time? $O(b^s)$
    - space? $O(b^s)$
  - Uniform Cost Search (UCS)
    - idea: lowest cost expand
    - implementation: heap PQ, use "backward cost"
    - complete? YES
    - optimal? YES, if nonnegative costs
    - time? $O(b^{C^*/\varepsilon})$ // C^*: optimal cost; ε: min cost between nodes
    - space? $O(b^{C^*/\varepsilon})$
- Informed Search
  - idea: use info about goal
  - Heuristics
    - input: state, output: estimate cost to goal
    - [illegible] lower bound, usually solution to relaxed problem
    - ex. Manhattan distance [illegible]
  - Greedy Search
    - idea: expand lowest heuristic value
    - implementation: PQ, like UCS, but estimated forward cost
    - complete? no } highly depends, varies a lot
    - optimal? no
  - A^* Search
    - idea: lowest estimated total cost
    - implementation: PQ, UCS + greedy
    - complete? YES } if good heuristic
    - optimal? YES
  - Admissibility and Consistency
    - $g(n)$ - backwards cost; UCS
    - $h(n)$ - estimate forward cost; greedy; $h^*(n)$ optimal
    - $f(n) = g(n) + h(n)$
    - Admissibility
      - needed for optimality
      - $0 \le h(n) \le h^*(n)$
      - Graph Search
        - idea: tree search but don't expand visited nodes
        - optimization - prevent extra computation
        - issue: makes A^* nonoptimal unless consistent
    - Consistency
      - $h(A) - h(C) \le$ ~~[illegible]~~ $cost(A, C)$
      - makes A^* graph search optimal
    - Dominance
      - if $h_a(n)$ dominates $h_b(n)$, $\Rightarrow \forall n: h_a(n) \ge h_b(n)$
      - intuition sense of better heuristics
      - taking max of multiple heuristics [illegible] admissible
        - good for making better heuristics

- Constraint Satisfaction Problems (CSPs)
  - identification problem - just need solution, no path
    - Variables - N vars, $X_1 \dots X_N$
    - Domain - set $\{x_1, \dots x_d\}$ each var can be
    - Constraints
  - NP-hard
    - N vars, $O(d)$ domain, $O(d^N)$ assignments
    - use partial assignments to make search problem
    - use heuristics too to solve reasonably
  - Constraint Types (Constraint Graphs)
    - unary - 1 var, prunes domains
    - binary - 2 var, edges
    - higher-order - 3+ var, [illegible] edges
  - Backtracking Search
    - idea: modified DFS
      - fix order for vars, like $X_1, X_2, \dots$
      - select value for var, if none go back
      - if none, backtrack, pick another
    - general alg for solving CSPs
    - Filtering
      - idea: prune unassigned var domains by removing vals we know will backtrack
      - Forward Checking
        - val assigned ⇒ prune var that share constraint
      - Arc Consistency
        - each undirected edge = 2 direct arcs
        - Algorithm ← of unassigned
          - store all arcs in queue Q
          - remove A→B; if var in A domain has no B var, remove it (tail → A, head → B)
          - if ≥1 remove from A, add all X→A (unassigned) to Q (unless already there)
          - if Q empty, stop; if domain empty, backtrack
      - AC-3
        - basically ↑
        - $O(ed^3)$ ← basically binary + higher-order constraints
          - e: # arcs / directed edges
          - d: size of largest domain
      - k-consistency: any set of k nodes ⇒ consistent assignment to k-1 nodes guarantees k^{th} node has ≥1 consistent value
        - strong k-consistency: also k, k-1, k-2, … consistent
        - arc consistency == 2-consistency
    - Ordering
      - don't use fixed ordering
      - Minimum Remaining Values (MRV) ← how to pick var
        - fewest valid values / most constrained
      - Least Constraining Value (LCV)
        - how to pick value
        - value that prunes fewest values
        - takes extra computation
    - Structure
      - tree structured CSP: $O(nd^2)$
        - pick root; $Parent(X_i) \to X_i$
        - topological sort
        - backwards pass of arc consistency
        - perform forward assignment
      - cutset conditioning
        - remove assign cutset, make tree
        - $O(d^c(n-c)d^2)$
          - c: size of smallest cutset
- Local Search
  - random assignment
  - pick random conflict var, reassign min violates fewest constraints (min-conflicts heuristic)
  - surprisingly good, but incomplete & suboptimal
  - almost constant even for large N
  - critical value $R = \frac{\#\text{constraints}}{\#\text{variables}}$ that [illegible]



- Bayes Net (Sampling)
  - Prior Sampling - make samples, calc based on that
  - Rejection Sampling - stop if e violated
  - Likelihood Weighting - never reject a sample
    - force e to be set
    - w = 1
      for i = 1 … n
        if X_i is evidence var e:
          X_i = evidence value
          $w = w \cdot P(x_i \mid Parent(X_i))$
        else
          x_i = sample from $P(X_i \mid Parent(X_i))$
      return $x_1, \dots, x_n, w$
    - to get prob, use w as a weight, normalize ← $\frac{\sum w}{\#\text{"whole" samples}}$
  - Gibbs Sampling
    - all vars random first, but fix evidence
    - resample one var conditioned on rest (not evidence)
    - repeat for long time

- Adversarial Search Problems (Games)
  - Looking at deterministic zero sum games
  - Minimax
    - opponent behaves optimally
    - terminal utilities
    - state value
    - $\forall$ max, $V(s) = \max_{s' \in successors} V(s')$
    - $\forall$ min, $V(s) = \min_{s' \in successors} V(s')$
    - $\forall$ terminal, $V(s)$ = known
    - postorder traversal
    - slow, $O(b^m)$
  - Alpha-Beta Pruning
    - α: MAX's best option
    - β: MIN's best option
    - note: if terminal or max depth, return utility of state / estimate

```
def maxV(s, α, β):
    v = -∞
    for each successor of s:
        v = max(v, value(succ, α, β))
        if v ≥ β: return v  // prune
        α = max(α, v)
    return v

def minV(s, α, β):
    v = +∞
    for each successor of s:
        v = min(v, value(succ, α, β))
        if v ≤ α: return v  // prune
        β = min(β, v)
    return v
```

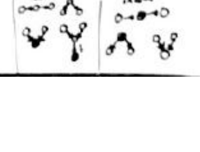
  - Evaluation Function
    - guess of utility of state
  - Expectimax
    - $\forall$ chance $V(s) = \sum_{s' \in succ} P(s' \mid s) V(s')$
    - and minimax
  - Mixed Layer Types
    - mix of above
  - General Games (not zero-sum)
    - multi-agent utilities (tuple of utilities)
  - Utilities
    - Principle of Maximum Utility
    - rational preferences
    - $A \succ B$ // A prefer to B
    - $A \sim B$ // don't care
    - lottery // probability
    - Axioms of Rationality
      - Orderability $(A \succ B) \lor (A \prec B) \lor (A \sim B)$
      - Transitivity $(A \succ B) \land (B \succ C) \Rightarrow A \succ C$
      - 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 $A \succ B \Rightarrow (p \ge q \Leftrightarrow [p, A; 1-p, B] \succeq [q, A; 1-q, B])$
    - Utility Function U
      - $U(A) \ge U(B) \Leftrightarrow A \succeq B$
      - $U([p_1, S_1; \dots p_n, S_n]) \Leftrightarrow \sum p_i U(S_i)$
      - risk-neutral / averse / seeking

- Non-Deterministic Search (Markov Decision Processes (MDPs))
  - MDP formulation
    - S: set of states
    - A: set of actions
    - start states
    - terminal states
    - γ: discount factor
    - $T(s, a, s')$ // prob
    - $R(s, a, s')$ // reward
  - q-state / action state - like expectimax chance node
  - γ - prevent infinite reach, geometric series $\le \frac{R_{max}}{1-\gamma}$
  - Markov/memoryless property
    - knowing present, past & future independent
    - $T(s, a, s') = P(s' \mid s, a)$
  - Bellman Equation (DP?)
    - $V^*(s)$: optimal value of s // 0 for terminal state
    - $Q^*(s, a)$: optimal value of q-state
    - $V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$
    - $Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$
    - $V^*(s) = \max_a Q^*(s, a)$
  - Value Iteration (DP alg?)
    - convergence $V_{k+1}(s) = V_k(s) \Rightarrow V_{k+1}(s) = V^*(s)$
    - $V_0(s) = 0, \forall s$ // init
    - $\forall s \in S$, $V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$
  - Policy Extraction
    - $\forall s \in S$, $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a) = \operatorname{argmax}_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$
  - Policy Iteration
    - any initial policy // closer = faster
    - solve system $V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$
    - $\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$
    - $\pi_{i+1} = \pi_i$ ⇒ convergence

- Reinforcement Learning (RL)
  - Online learning / exploration
  - Model Based Learning
    - $\hat{T}(s, a, s')$ // estimate func by samples $\frac{\#\text{[illegible]}}{\#\text{[illegible]}}$
    - $\hat{R}(s, a, s')$ [illegible]
    - Law of Large Numbers, works pretty well
    - use policy iteration after
  - Model Free Learning
    - passive RL / on-policy
      - Direct Evaluation
        - fixed policy π
        - eventually learns $V^{\pi}(s)$
      - Temporal Difference (TD)
        - $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$
        - $V^{\pi}(s) = (1-\alpha)V^{\pi}(s) + \alpha \cdot sample$
        - α: learning rate, decrease over time
    - active RL / off-policy
      - Q-learning
        - q-value iteration ideas like value iteration
        - $sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
        - $Q(s, a) = (1-\alpha) Q(s, a) + \alpha \cdot sample$
        - α: learning rate, decrease over time
      - Approximate Q-learning
        - S too big ⇒ use feature vector
        - linear value function
        - $difference = [R(s, a, s') + \gamma \max_{a'} Q(s', a')] - Q(s, a)$
        - $w_i = w_i + difference \cdot \alpha \cdot f_i(s, a)$
        - note: Q-learning [illegible] $Q(s, a) = Q(s, a) + \alpha \cdot difference$
  - Exploration and Exploitation
    - ε-Greedy Policies
      - Prob ε to be random
      - too small = slow learn, need to lower over time
    - Exploration Functions
      - don't need to lower ε
      - $Q(s, a) = (1-\alpha)Q(s, a) + \alpha \cdot [R(s, a, s') + \gamma \max_{a'} f(s', a')]$
      - $f(s, a)$: exploration function, prefer to do things haven't done, rewards unexplored
      - ex. $f(s, a) = Q(s, a) + \frac{k}{N(s, a)}$ ← # times done

- Probabilistic Inference
  - Joint distribution - all probs } slow
  - Inference by enumeration (IBE)
    - $P(Q_1 \dots Q_k \mid e_1, \dots e_k)$
    - Query Var Q_i: [illegible]
    - Evidence Var E_i: [illegible], known
    - Hidden Vars: not there
    - collect & sum out
  - Bayes Net (Representation)
    - compact, uses local PDF, DAG
    - each node is conditionally independent of all ancestor nodes, given parents
    - store $P(X \mid Parents(X))$
    - basically DAG + conditional probability tables (CPT)
    - $P(A \mid B) = \frac{P(B \mid A) P(A)}{P(B)}$
  - Bayes Net (Inference)
    - inference - calc joint PDF for Q, e
    - eliminate var X
      - join / multiply all factors w/ X
      - sum out X → [illegible]
  - Bayes Net (D-separation)
    - Causal Chain
      - $X \to Y \to Z$
      - $X \perp\!\!\!\perp Z \mid Y$ // given Y, X Z independent
    - Common Cause
      - $X \leftarrow Y \to Z$ $X \perp\!\!\!\perp Z \mid Y$
    - Common Effect
      - $X \to Y \leftarrow Z$ X Z independent if Y unknown
    - D(Direct)-separation
      - $X \perp\!\!\!\perp Y \mid \{Z_1, \dots Z_k\}$
      - alg
        - shade $Z_1, \dots Z_k$
        - find all undirected paths X to Y
        - each path:
          - break into triples
          - if all active, dependent
          - if none, then independent

active   inactive



Matthew Tran
CS188
Final

- Markov Models
  - chain ⇒ memoryless property
    - $P(W_0, \dots W_n) = P(W_0) \prod P(W_{i+1}|W_i)$
    - assume stationary :: same
  - Mini-Forward Algorithm
    - $P(W_{i+1}) = \sum_{w_i} P(W_{i+1}|w_i) P(w_i)$
  - Stationary Distribution
    $P(t_{+1}) = P(t)$

- Hidden Markov Model
  - collect evidence that affects beliefs
  - state var, evidence var
  - assume $P(F_i|W_i)$ stationary
  - only know evidence
  - Forward Algorithm
    - $B(W_i) = P(W_i | F_1 \dots F_i)$
    - $B'(W_i) = P(W_i | F_1 \dots F_{i-1})$
    - $B'(W_{i+1}) = \sum_{w_i} P(W_{i+1}|w_i) B(w_i)$
    - $B(W_{i+1}) \propto P(F_{i+1}|W_{i+1}) B'(W_{i+1})$
      - normalize to get new belief
    - $B(W_{i+1}) \propto P(F_{i+1}|W_{i+1}) \sum_{w_i} P(W_{i+1}|w_i) B(w_i)$
  - Viterbi Algorithm
    - most likely sequence of states given evidence
    - forward and backward pass
  - Particle Filtering
    - n particles, d states, $n \ll d$
    - Time Elapse Update
      - sample new particle from dist
    - Observation Update
      1. weight each particle w/ $P(F_i|T_i)$
      2. get total weight each state
      3. if sum all weights all states = 0, resample all
      4. else resample based on weight dist

- Decision Networks
  - chance, action, utility nodes
  - Maximum Expected Utility (MEU)
    - $EU(a|e) = \sum_{x_1 \dots x_n} P(x_1 \dots x_n|e) U(a, x_1 \dots x_n)$
    - argmax over EU, basically expectimax + Bayes
  - Value of Perfect Information (VPI) - (new evidence value)
    - $MEU(e) = \max_a \sum_s P(s|e) U(s,a)$
    - $MEU(e,e') = \max_a \sum_s P(s|e,e') U(s,a)$
    - $MEU(e,E') = \sum_{e'} P(e'|e) MEU(e,e')$
    - $VPI(E'|e) = MEU(e,E') - MEU(e)$
    - Properties
      - nonnegativity, $VPI \geq 0$
      - nonadditivity, in general
      - order-independence

- Machine Learning
  - dataset: training, validation, test
  - Naive Bayes - classification
    - feature vector ⇒ label, assume features independent ⊥ label
  - Parameter Estimation
    - Maximum Likelihood Estimation (MLE)
      - assume identically distributed, independent
      - assume all parameter values equal prob, θ
      - $\mathcal{L}(\theta) = \prod_{i=1}^{n} P_\theta(x_i)$ return $\frac{\partial}{\partial \theta} \mathcal{L}(\theta) = 0$
  - Laplace Smoothing - k
    - avoid overfitting
    - assume k extra of each possibility
    - $k \to \infty$ == all equal prob
  - Perceptron
    - Linear Classifier
      - activation = linear combo of features (dot product)
    - Binary Perceptron
      - 2 choices, hyperplane
      - all weights = 0
      - if activation = $w^T f > 0$, +1, else -1 = y (y^* = true label)
      - if $y \neq y^*$, $w = w + y^* f(x)$
      - if no update, exit
    - Bias
      - shift hyperplane from origin
      - just add feature of value 1 and it works
    - Multiclass Perceptron
      - if wrong, add to correct weight
        subtract from wrong weight

- Neural Networks
  - Motivation
    - nonlinear separators
  - Idea - multilayer perceptron
    - universal function approximator
  - Multilayer Feedforward Neural Networks
    - idea - apply diff nonlinearity
    - perceptron - use step
    - can use ReLU
    - sigmoid $\frac{1}{1+e^{-x}}$
  - Gradient Ascent/Descent
    - $\nabla_w ll_w = \left[\frac{\partial ll(w)}{\partial w_1} \dots \right]$ (ll = loss)
    - $w \leftarrow w + \alpha \cdot \nabla_w ll(w)$ (α = learning rate)
    - batch gradient descent - use all points (slow)
    - mini-batching - take batches
    - stochastic gradient descent (SGD) - k=1
  - Backpropagation
    - how to calc gradients
    - forward pass - normal calc
    - backward pass - get gradients