CS 188 Midterm Notes

Search Problems

- state space size $= \prod_{i=1}^{n} x_i$, where variable x_i has i possible values
- expanding a frontier = replacing length n plan with all len $n+1$ plans stemming from it.
- completeness = if solution exists, will it eventually be found?
- optimality = is the solution a cost minimizer.
- branching factor $= b =$ number of children of a node. $O(b^k)$ nodes at depth k.
- tree search revisits nodes, graph search doesn't. [s = depth of shallowest solution] [m = max depth] [C^* = optimal cost] [ε = min. cost b/w 2 nodes]

BFS: complete, non-optimal except if edge costs equivalent, $O(b^s)$ time + space

DFS (tree): $O(b^m)$ time, $O(mb)$ space, non-complete, non-optimal

Uniform Cost Search: complete, queues based on backward cost: cost(S, n), $O(b^{C^*/\varepsilon})$ time + space, optimal if $\varepsilon \geq 0$.

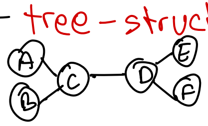
- Heuristic: estimation of forward cost, cost(n, G). — tree search + visited set = graph search
- Consistency: $h(b) - h(a) \leq$ cost(a,b) $\forall a,c$ — admissibility: $h(n) \leq$ cost(n, G) $\forall n$.
- Dominance: $h_1(n) \geq h_2(n)$ $\forall n$. [h_1 is dominant over h_2].
- A^* algorithm: uses $f(n) = \text{cost}(S,n) + h(n)$ to queue states.
- A^* tree search is optimal for admissible heuristics, but not complete.
- A^* graph search is complete, but only optimal for consistent heuristics.

graph search has visited set

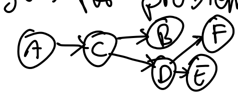
Greedy: only considers heuristic and not backward cost.
- unpredictable but quick.
- non-optimal, non-complete

Constraint Satisfaction Problems

- CSPs are NP-hard. $O(d^N)$ [d = domain of 1 var, N = # of var]
- Back-tracking: with variables (x_i), assign a value for x_k, satisfying all constraints wrt $x_1 \ldots x_{k-1}$. if no value exists, go back to x_{k-1} and re-assign.
- forward checking: type of filtering. when assigning value, prune domains of unassigned variables.
- arc consistency: store all constraints in a queue as directed arcs. while checking $A \to B$, remove a from domain(A) if it fails for all values $b \in$ domain(B). if domain(A) changes, queue $(X \to A)$ $\forall X$.
  also AC-3, time: $O(ed^3)$ (e = # arcs, d = domain size)
- k-consistency: for any k nodes, assigning any $k-1$ nodes leaves at least 1 consistent value for k^{th} node. [strong k-consistency guarantees i-consistency $\forall i \in [k]$]
- Minimum Remaining Values: when selecting variable to assign next, pick most constrained variable.
- Least Constraining Value: when selecting value to assign, run forward checking/arc consistency, pick value that prunes the least.
- tree-structured algo: for problems with acyclic constraint graph,



step 1: linearize



step 2: Backward pass: Enforce Parent$(X_i) \to X_i$ $\forall i$ → step 3: forward assignment: assign variables forward. $O(Nd^2)$.

- Cutset conditioning: find cutset (smallest subset of variables, removing which yields tree) assign values to cutset, prune the remaining variables as a tree CSP. backtrack if no solution, reassign cutset. [backtracking upto d^c times].
  (let c = |cutset+1|) $O(d^c (n-c) d^2)$
- Local Search: randomly assign, randomly select conflicted variable, reassign using min-conflicts heuristic. incomplete, suboptimal, but fast in most cases
  - hill climbing (choose best successor until no improving successors, even if local maxima).
  - simulated annealing (choose successor if improvement or w/ prob $e^{\text{diff}/T}$ if not; lower T slowly enough for optimal and complete)
  - genetic algos (similar with population metaphor)

Games

- Minimax: $\forall s \in$ Agent controlled, $V(s) = \max_{s' \in \text{successors}(s)} V(s')$; $\forall s \in$ opp. controlled, $V(s) = \min_{s' \in \text{succ}(s)} V(s')$ (DFS on game tree)
- Alpha Beta Pruning: α: MAX's, β: MIN's best option on path to root

def max-value(s, α, β): $v = -\infty$ $\forall s' \in$ succ(s): $v = \max(v,$ value$(s', \alpha, \beta))$ if $v \geq \beta$ return v $\alpha = \max(\alpha, v)$ return v	def min-value(s, α, β): $v = \infty$ $\forall s' \in$ succ(s): $v = \min(v,$ value$(s', \alpha, \beta))$ if $v \leq \alpha$ return v $\beta = \min(\beta, v)$ return v

Evaluation functions:
- In depth limited game trees, terminal nodes are non-terminal game states
- $V(T) = \sum_i w_i f_i(T)$, f is feature

- Expectimax: $\forall s \in$ opp. controlled, $V(s) = \sum_{s' \in \text{succ}(s)} P(s'|s) V(s')$
- Monte-Carlo Tree Search: high branching factor → tree pruning difficult. Play each move many times, count wins.
  ($\lim_{N \to \infty}$, MCTS UCT → minimax).
- MCTS UCT algo: use UCB criterion to go down tree until uncharted node. play move from node, record win(?) and go back to root.
  (winning action is $\arg\max_n N(n)$)

$$UCB(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{\log N(\text{Parent}(n))}{N(n)}}$$

$N(n)$ = # of rollouts of n
$U(n)$ = # of wins with n
C = hyperparam. higher C = more exploration

Markov Decision Processes

— $U(\text{Path}) = \sum_{i=0}^{n} \gamma^i R(s_i, a_i, s_{i+1})$ — $\lim_{n\to\infty} U(\text{Path}) \le \frac{R_{max}}{1-\gamma}$ if $|\gamma|<1$.

— **Value Iteration**: Initialise $\vec{V_0} = \vec{0}$, then iterate to convergence, then extract policy

|A| actions, |S| states

— Bellman Update: $\forall s\ U_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma U_k(s'))$

$O(|S|^2|A|)$ — Bellman Equations: $Q^*(s,a) = \sum_{s} T(s,a,s')(R(s,a,s') + \gamma U^*(s'))$

— Policy Extraction: $\pi^*(s) = \arg\max_a Q^*(s,a)$

— **Policy Iteration**: fix initial policy, then loop till $\pi_{i+1} = \pi_i\ [=\pi^*]$:

$O(|S|^3)$ — Policy Evaluation: $U^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma U^\pi(s')]$

— Policy Improvement: $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^{\pi_i}(s')]$

main diff: instead of considering all actions, consider a random one and then iterate.

Reinforcement Learning

— **Model-based**: use explored samples to estimate T & R, then use value / policy iteration.

— $\hat{T}(s,a,s') = \frac{\#\ s \xrightarrow{a} s'}{\#\ s \xrightarrow{a}}$ — $\hat{R}(s,a,s') = \frac{\sum_i R_i(s,a,s')}{\#\ s \xrightarrow{a} s'}$

— **Model-free**:

— Passive RL:

— **Direct Evaluation**: start in different states, play to terminal for each, average (total reward) U values found.

— **Temporal Difference**: sample $= R(s,\pi(s),s') + \gamma V^\pi(s)$, α = learning rate

update: $V^\pi(s) = (1-\alpha)V^\pi(s) + \alpha \cdot \text{sample} = V^\pi(s) + \alpha(R(s,\pi(s),s') + (\gamma-1)V^\pi(s))$

[$V^\pi(s) = \alpha \sum_{i=1}^{k} (1-\alpha)^{k-i} \text{sample}_i$, so older samples given exponentially less weight]

— Active RL: **Q-Learning**: value iteration Q instead of U. [aka V]; $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \text{sample}$

— Q-learning is off-policy, i.e. it can derive optimal policy even with suboptimal moves.

— **Approximate Q-Learning**: uses features and weights to learn patterns

— $V(s) = \vec{\omega}^T \cdot \vec{f}(s)$ $Q(s,a) = \vec{\omega}^T \cdot \vec{f}(s,a)$

— difference $= [R(s,a,s') + \gamma \max_{a'} Q(s',a')] - Q(s,a)$

— update: $\vec{\omega} \mathrel{+}= \alpha \cdot \text{difference} \cdot \vec{f}(s,a)$ (same as Q-learning, where $Q(s,a) \mathrel{+}= \alpha\,\text{difference}$)

— better for more states, better to understand the game.

— **ε-greedy policy**: explores with prob. ε, exploits with 1−ε.
consistently high/low ε = slow convergence. tune manually (high to low slowly)

— **exploration fns**: $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(R(s,a,s') + \gamma \max_{a'} f(s,a'))$,
where f is exploration fn. like $Q(s,a) + \frac{K}{N(s,a)}$

— **regret**: Total reward acting optimal — Reward from learning algo.

Bayesian Nets

— N variables, represent relationships in directed acyclic graph.

— For each node X, store $P(X|A_1..A_n)$, where A_i are parents of X, in **Conditional Prob Table**.

— $\mathbb{P}[\text{Path}] = \prod_{X \in \text{Path}} \mathbb{P}[X|\text{Parents}(X)]$ — Markov property: X cond. indep. of ancestors given parents

O→O→O causal chain,  common cause,  common effect.

— $\{Z_1,..Z_k\}$ **d-separates** X, Y $\Rightarrow X \perp\!\!\!\perp Y | \{Z_1,..Z_k\}$

— d-separation algo checks for d-separation:

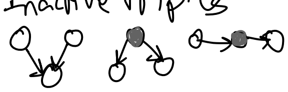
— shade all observed $\{Z_1..Z_k\}$ — enumerate all paths $X \to Y$.

— decompose each path into triples. if all triples active, the path d-connects.

— if no path d-connects X to Y, they are d-separated

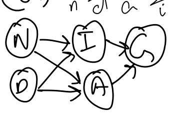
— Active triples

— Inactive triples



— Inference by Enumeration

$P(G) = \sum_n \sum_d \sum_a \sum_i P(n)P(d)P(i|d,n)P(a|d,n)P(G|a,i)$



[Requires creation of exponentially large CPT]

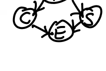
— Inference via Variable Elimination

— multiply all factors involving X.

— sum out X.

factors ← []
for each *var* **in** ORDER(*bn*.VARS) **do**
factors ← [MAKE-FACTOR(*var*, **e**)|*factors*]
if *var* is a hidden variable **then** *factors* ← SUM-OUT(*var*, *factors*)
return NORMALIZE(POINTWISE-PRODUCT(*factors*))

observed values for variable E.

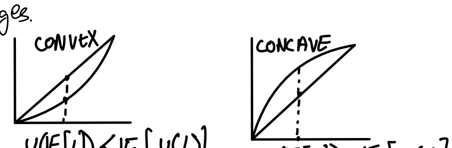
$[P(T|e) = \alpha P(T) \sum_s P(s|T) \sum_c P(c|T) P(e|c,s)$ for 

- Sampling [Approximate inference for BNs]
  - Prior sampling: generate samples of [J, C]. use to calculate $\mathbb{P}[C|T]$.
  - Rejection sampling: early reject bad samples. ex: for $\mathbb{P}[C|T=-t]$, throw away any samples as soon as $T \neq -t$. (don't generate C for samples with $T=t$).
  - Likelihood weighting: [start all sample weights at 1]
    - for evidence variables, fix value and multiply weight of sample by $\mathbb{P}[E=e|\text{Parents}(E)]$.
    - for all other variables, sample value.
  - Gibbs sampling:
    - prerequisite: CPTs of all variables w.r.t. neighbours. If most nodes have few neighbors, computable in linear time.
    - set all non-evidence variables to some random value, clear one var. at a time and sample it from its CPT w.r.t. all other vars.
    - estimate for $P(X|e)$ is bad at first, but eventually converges.

function GIBBS-ASK(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  local variables: **N**, a vector of counts for each value of X, initially zero
    Z, the nonevidence variables in bn
    x, the current state of the network, initially copied from **e**

  initialize **x** with random values for the variables in **Z**
  for $j = 1$ to N **do**
    for each Z_i in **Z** **do**
      set the value of Z_i in **x** by sampling from $\mathbf{P}(Z_i|mb(Z_i))$
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in **x**
  return NORMALIZE(**N**)

Figure 14.16 The Gibbs sampling algorithm for approximate inference in Bayesian networks; this version cycles through the variables, but choosing variables at random also works.



$U(E[L]) < E[U(L)]$ $\quad$ $U(E[L]) > E[U(L)]$

- Convex Utility function $\Leftrightarrow$ Risk-seeking agent.
- Concave Utility function $\Leftrightarrow$ Risk-averse agent.

Chance Nodes - random variables like BNs.

Action Nodes - agent can choose action here.

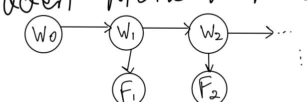
Utility Nodes - assign a utility value here

Decision Networks

- An agent has rational preferences iff:
  - $(A \succ B) \vee (A \prec B) \vee (A \sim B)$ $\quad$ - $(A \sim B) \Rightarrow [p, A; (1-p), C] \sim [p, B; (1-p), C]$ $\quad$ - $(A \succ B) \Rightarrow \{(p \geq q) \Leftrightarrow ([p, A; (1-p) B] \succeq [q, A; (1-q) B])\}$
  - $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$ $\quad$ - $(A \succ B \succ C) \Rightarrow \exists p [p, A; (1-p), C] \sim B$
- Value of Perfect Information: $VPI(E'|e) = MEU(e, E') - MEU(e)$ is value of observing E' given e already.
  - $\forall E', e : VPI(E'|e) \geq 0$ $\quad$ - $VPI(E_k, E_j|e) = VPI(E_j|e) + VPI(E_k|E_j, e)$
    $= VPI(E_k|e) + VPI(E_j|E_k, e)$.

Markov Models

- characterized by transition probabilities, initial distribution [assume memorylessness].
- $P[W_0 \dots W_n] = P(W_0) \prod_{i=0}^{n-1} P(W_{i+1}|W_i)$. $\quad$ - transition models are usually stationary $[P(W_{i+1}|W_i)$ identical $\forall i]$
- To marginalize $P(W_{i+1})$, use mini-forward algo: $P(W_{i+1}) = \sum_{w_i} P(w_i) \cdot P(W_{i+1}|w_i)$
- To solve for stationary dist. use $P(W_i) = \sum_{w_i} P(w_i) P(W_{i+1}|w_i)$. (for all W_i).
- Hidden Markov Models assume both $\underbrace{P(W_{i+1}|W_i)}_{\text{transition model}}$ and $\underbrace{P(F_i|W_i)}_{\text{sensor model.}}$ are stationary



- Belief Distributions: $B(W_i) = P(W_i|f_1, \dots, i)$, $B'(W_i) = P(W_i|f_1, \dots, i-1)$. and $\sum_{w_i} B(w_i) = 1$.
  $\Rightarrow B'(W_{i+1}) = \sum_{w_i} P(W_{i+1}|w_i) B(w_i)$, $\quad B(W_{i+1}) \propto P(f_{i+1}|W_{i+1}) B'(W_{i+1})$
- Forward algorithm: $B(W_{i+1}) \propto P(f_{i+1}|W_{i+1}) \sum_{w_i} P(W_{i+1}|w_i) B(w_i)$ [$B(W_0)$ is initial dist. $P(W_0)$]

$O(dn)$ - time-elapse update: use $B(W_i)$ to find $B'(W_{i+1}) \rightarrow$ observation update: observe f_i, find $B(W_{i+1})$ (and normalize) $\rightarrow i{+}{=}1$, repeat.

- Viterbi algorithm:

$O(d^2 n)$

Result: Most likely sequence of hidden states $x^*_{1:N}$
```
/* Forward pass                                                    */
for t = 1 to N do
    for x_t ∈ 𝒳 do
        if t = 1 then
        |   m_t[x_t] = P(x_t)P(e_0|x_t)
        else
            a_t[x_t] = argmax_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}];
            m_t[x_t] = P(e_t|x_t) P(x_t|a_t[x_t]) m_{t-1}[a_t[x_t]];
        end
    end
end
/* Find the most likely path's ending point                        */
x*_N = argmax_{x_N} m_N[x_N];
/* Work backwards through our most likely path and find the hidden
   states                                                          */
for t = N to 2 do
|   x*_{t-1} = a_t[x*_t];
end
```

- Particle Filtering: approximate inference for HMMs
  - Initialization: random / uniform / initial dist.
  - Time-elapse: sample new states from $P(W_{t+1}|w_t)$
  - Observation:
    - weight particles with $P(F_{t+1}|W_{t+1})$
    - calculate sum of weights for each state
    - if sum of weights across all states is 0, re-init
    - else, normalize dist. of weights across states.

ML

- $P_{MLE}(x) = \frac{count(x)}{N}$, Laplace smoothing: $P_{LAP,k}(x) = \frac{count(x) + k}{N + k|X|} \Rightarrow P_{LAP,\infty}(x) = \frac{1}{|X|}$
  $\Rightarrow P_{LAP,k}(x|y) = \frac{count(x, y) + k}{count(y) + k|X|}$ $\therefore$ smooths toward uniform.
- Multi-class perceptron: k classes, f features, $W \in \mathbb{R}^{k \times f}$
  - one epoch: classify every point. [i^{th} row of W].
    (if x_i is misclassified, $(\forall j : j \neq y)$ $\vec{w}_j$ -= $\vec{x}_i$, $\vec{w}_y$ += $\vec{x}_i$.
- softmax: $P(y=i|\vec{x}, W) = \frac{e^{\vec{w}_i^T \vec{x}}}{\sum_j e^{\vec{w}_j^T \vec{x}}}$. - Logistic Regression: $h_{\vec{w}}(\vec{x}) = \frac{1}{1+e^{-\vec{w}^T \vec{x}}}$.