In the name of God, the Compassionate, the Merciful

ARTIFICIAL INTELLIGENCE

Chapter 3: Search

A **strategy** is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
- **Completeness:** Does it always find a solution if one exists?
- **Time complexity:** Number of nodes generated/expanded
- **Space complexity:** Maximum number of nodes in memory
- **Optimality:** Does it always find a least-cost solution?

Time and space complexity are measured in terms of (Dr.)
- **b:** Maximum branching factor of the search tree
- **d:** Depth of the least-cost solution
- **m:** Maximum depth of the state space (may be ∞)

Search Types:
- **Uninformed:** The agent has no information about the underlying problem other than its definition. e.g. Breadth-first, Uniform-cost, Depth-first, Depth-limited, Iterative deepening
- **Informed:** The agent have some idea of where to look for solutions

Tree search algorithms: offline, simulated exploration of state space by generating successors of already-explored states
- Breadth-first search(BFS):Expand shallowest unexpanded node.
- Uniform-cost search(UCS): Expand least-cost unexpanded node.
- Depth-first search(DFS): Expand deepest unexpanded node
- Depth-limited search(DLS): depth-first search with depth limit l
- Iterative deepening search(IDS):

Criterion:	Complete?	Time	Space	Optimal?
BFS	Yes[1]	O(b^{d+1})	O(b^{d+1})	Yes[2]
UCS	Yes[3]	O($b^{\lceil \frac{c^*}{\epsilon} \rceil}$)	O($b^{\lceil \frac{c^*}{\epsilon} \rceil}$)	Yes[4]
DFS	No[5]	O(b^m)	O(bm)	No
DLS	No	O(b^l)	O(bl)	No
IDS	Yes	O(b^d)	O(bd)	Yes

1. if b is finite
2. if cost = 1 per step
3. if step cost $\geq \epsilon$
4. nodes expanded in increasing order of $g(n)$
5. fails in infinite-depth spaces

Chapter 4: Informed search

Best-first search: Expand most desirable unexpanded node
Special cases: greedy search, A* search

Greedy best-first search: expands the node that appears to be closest to goal

A^* **Search:** avoid expanding paths that are already expensive
Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = heuristic, estimated cost to goal from n
- $f(n)$ = estimated total cost of path through n to goal
- A^* search uses an **admissible** heuristic i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true cost** from n.

Criterion:	Complete?	Time	Space	Optimal?
Greedy	No[1]	O(b^m)	O(b^m)	No
A^*	Yes	exp	all nodes	Yes

1. can get stuck in loops.

Local search algorithms
- When path doesn't matter (e.g., optimization problems).
- keep a single "current" state, try to improve it.
- **Two key advantages:**
  1. they use very little memory usually a constant amount.
  2. they can often find reasonable solutions in large or infinite (continuous) state spaces

Hill-climbing search problem: 1- Local maximum 2- Plateau (resulting in a random walk) 3- Ridges (slopes very gently toward a peak)
SOLUTION: Random restart hill-climbing

Simulated Annealing:
- Escape local maxima by allowing some "bad" moves but gradually decrease their **size** and **frequency**.
- Probability of a move decreases with the amount ΔE by which the evaluation is worsened.
- high T allows more worse moves, T close to zero results in few or no bad moves.
- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.
- Widely used in VLSI layout, airline scheduling, etc.

Local beam search:
- Keep track of k states rather than just one.
- Start with k randomly generated states.
- At each iteration, all the successors of all k states are generated.
- If any one is a goal state, stop; else select the k best successors from the complete list and repeat.

Genetic algorithms
- A successor state is generated by combining two parent states.
- Start with k randomly generated states (**population**).
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s).
- Evaluation function (**fitness function**): Higher values for better states.
- Produce the next generation of states by **selection**, **crossover**, and **mutation**.

Chapter 5: Adversarial Search (Games)

Minimax
- Perfect play for deterministic games
- Idea: choose move to position with highest **minimax value** = best achievable payoff against best play

Criterion:	Complete?	Time	Space	Optimal?
Minimax	Yes[1]	O(b^m)	O(bm)	Yes[2]

1. if tree is finite
2. against an optimal opponent

$\alpha - \beta$ **pruning**
- α: the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
- β: the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.
- Alphabeta search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to **be worse than** the current α or β value **for MAX or MIN**, **respectively**.

Algorithm

1. Search down the tree to the given depth (Depth first Search).
2. Once reaching the bottom, calculate the evaluation for this node.(i.e. it's utility).
3. Backtrack, propagating values and paths (according to what shown in next slides).
4. When the search is complete, the Alpha value at the top node gives the minimum score that the player is guaranteed to attain if using the path stored at the top node.

Properties of $\alpha - \beta$ pruning

- Pruning does not affect final result
- With "perfect ordering," time complexity = $O(b^{m/2})$

Chapter 6: CSP

Standard search formulation (incremental)

- **Initial state**: the empty assignment
- **Successor function**: assign a value to an unassigned variable that does not conflict with current assignment
- **Goal test**: the current assignment is complete.

Backtracking search for CSPs Backtracking search

- Depth-first search for CSPs with single-variable assignments is called backtracking search.
- Backtracking search is the basic uninformed algorithm for CSPs.

How to Improve

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failure early?

✓ **Minimum remaining values (MRV)**: choose the variable with the fewest legal values.
✓ **Degree Heuristic: Most Constraining Variable (MCS)**: choose the variable with the most constraints on remaining variables.
✓ **Least Constraining Value (LCV)**: To choose a value which puts minimum constraint on other variables.

Forward checking prevents assignments that guarantee later failure for $YinX \rightarrow Y$
Maintaining Arc Consistency start with only the arcs (Xj,Xi) for all Xj that are unassigned variables that are neighbors of X by AC-3. **Constraint propagation** (e.g., **arc consistency**) does additional work to constrain values and detect inconsistencies.

Chapter 14: Bayes Net

Probability Formula:

- **Chain Rule:**

$$P(x_1,x_2,...,x_n) = \prod_{i=1}^{n} P(x_i|x_{i-1},...,x_1) = \prod_{i=1}^{n} P(X_i|Parent(X_i))$$

Exactly Inference:
Enumeration Ask: $P(X|e) = \alpha P(X|e) = \alpha \sum_y P(X,e,y)$
Variable Elimination:

- **Query:** $P(Q|e_{1:k})$
- **Start with initial factors:** Local CPTs (If evidence, start with factors that select that evidence)
- **While there are still hidden variables (not Q or evidence):**
  - Pick a hidden variable H
  - Join all factors mentioning H
  - Eliminate (sum out) H
- **Join all remaining factors and normalize**

Approximately Inference:

- **Prior Sampling:** $S_{PS}(x_1,...,x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$
- **Rejection Sampling:** Use Prior Sapling and reject the x not consistent with evidence e.
- **Likelihood Weighting:**
  - $S_{WS}(z,e) = \prod_{i=1}^{l} P(z_i|parents(Z_i))$ (Z is not the evidence variable)
  - $w(z,e) = \prod_{i=1}^{m} P(e_i|parents(E_i))$ (likelihood weight)
  - Likelihood Weighting:

$$\begin{aligned} P(z,e) &= S_{WS}(z,e)w(z,e) \\ &= \prod_{i=1}^{l} P(z_i|parents(Z_i)) \prod_{i=1}^{m} P(e_i|parents(E_i)) \end{aligned}$$

- **Gibbs Sampling:**

$$P(x_i|mb(X_i)) = \alpha P(x_i'|parents(X_i)) \times \prod_{y_j \in Children(X_i)} P(y_j|parents(Y_j))$$

Chapter 15: Probabilistic over Time

Time and Uncertainty

- **Transition and sensor models:**
  - **Sensor model:** $P(E_t \mid X_t)$
  - **Transition model:** $P(X_t \mid X_{t1})$
  - **Markov assumption:**
    $$P(X_t \mid X_{0:t1}) = P(X_t \mid X_{t1})$$
  - **Sensor Markov assumption:**
    $$P(E_t \mid X_{0:t}, E_{0:t1}) = P(E_t \mid X_t)$$
  - **Complete joint distribution:**
    $$P(X_{0:t}, E_{1:t}) = P(X_0)\Pi_{i=1}^{t} P(X_i \mid X_{i-1})P(E_i \mid X_i)$$

Inference in Temporal Models

- **Filtering:**

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= P(X_{t+1}|e_{1:t}, e_{t+1}) \\ &= \alpha P(e_{t+1}|X_{t+1}, e_{1:t})P(X_{t+1}|e_{1:t}) \\ &= \alpha P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t}) \\ &= \alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t, e_{1:t})P(x_t|e_{1:t}) \\ &= \alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t)P(x_t|e_{1:t}) \end{aligned}$$

- **Prediction:**

$$P(X_{t+k+1}|e_{1:t}) = \sum_{x_{t+k}} P(X_{t+k+1}|x_{t+k})P(x_{t+k}|e_{1:t})$$

- **Smoothing:**

$$\begin{aligned} P(X_k|e_{1:t}) &= P(X_k|e_{1:k}, e_{k+1:t}) \\ &= \alpha P(X_k|e_{1:k})P(e_{k+1:t}|X_k, e_{1:k}) \\ &= \alpha P(X_k|e_{1:k})P(e_{k+1:t}|X_k) \\ &= \alpha f_{1:k} \times b_{k+1:t} \end{aligned}$$

$$\begin{aligned} P(e_{k+1:t}|X_k) &= \sum_{x_{k+1}} P(e_{k+1:t}|X_k, x_{k+1})P(x_{k+1}|X_k) \\ &= \sum_{x_{k+1}} P(e_{k+1:t}|x_{k+1})P(x_{k+1}|X_k) \\ &= \sum_{x_{k+1}} P(e_{k+1}, e_{k+2:t}|x_{k+1})P(x_{k+1}|X_k) \\ &= \sum_{x_{k+1}} P(e_{k+1}|x_{k+1})P(e_{k+2:t}|x_{k+1})P(x_{k+1}|X_k) \end{aligned}$$

$$b_{k+1:t} = \text{BACKWARD}(b_{k+2:t}, e_{k+1})$$

- **Finding the most likely sequence:**

$$\max P(x_1,...,x_t,X_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|X_{t+1}) \max_{x_t}[P(X_{t+1}|x_t)$$
$$\max_{x_1,\dots,x_{t-1}} P(x_1,...,x_{t-1},x_t|e_{1:t})]$$

- **Particle Filtering:**

$$N(x_t|e_{1:t})/N = P(x_t|e_{1:t})$$

  - **Transfer: Total number of samples reaching x_{t+1} is:**
    $$N(x_{t+1}|e_{1:t}) = \sum_{x_t} P(x_t+1|x_t)N(x_t|e_{1:t})$$
  - **Weight: Total weight of the samples in x_{t+1} after seeing e_{t+1} is:**
    $$W(x_{t+1}|e_{1:t+1}) = P(e_{t+1} \mid x_{t+1})N(x_{t+1|e_{1:t}})$$
  - **Resample: The number of samples in state x_{t+1} after resampling is proportional to the total weight in x_{t+1} before resampling:**
    $$N(x_{t+1}|e_{1:t+1})/N$$

Chapter 17: MDP

Sequential Decision Problems

- Utilities over time:
  - **Additive rewards:**
    $$U_h([s_0,s_1,s_2,\cdots]) = R(s_0)+R(s_1)+R(s_2)+\cdots$$
  - **Discounted rewards:** The discount factor γ is a number between 0 and 1.
    $$U_h([s_0,s_1,s_2,\cdots]) = R(s_0)+\gamma R(s_1)+\gamma^2 R(s_2)+\cdots$$
- Optimal policies and the utilities of states:
  - **Definition:** The expected utility obtained by executing π starting in s is given by:
    $$U^\pi(s) = E\big[\sum_{t=0}^{\infty} \gamma^t R(S_t)\big]$$
  - **Optimal policy:** An optimal policy when s is the starting state is:
    $$\pi_s^* = \arg\max_\pi U^\pi(s)$$
    $$\pi^*(s) = \arg\max_{a\in A(s)} \sum_{s'} P(s' \mid s,a)U(s')$$

Calculating an optimal policy

- Value Iteration:
  - **The Bellman equation for utilities:**
    $$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s,a)U(s')$$
  - **Bellman update:**
    $$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s' \mid s,a)U_i(s')$$
  - **If R is related with action:**
    $$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$
  - **Converge Condition**
    $$||U_{i+1} - U|| < \epsilon \rightarrow ||U^{\pi_i} - U|| < 2\epsilon\gamma/(1-\gamma)$$
- Policy Iteration:
  - **Simplified version of the Bellman equation:** It's relating the utility of s (under π_i) to the utilities of its neighbors:
    $$U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s,\pi_i(s))U_i(s')$$
  - **Simplified Bellman update/Modified policy iteration:**
    $$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s' \mid s,\pi_i(s))U_i(s')$$

Chapter 21: Reinforcement Learning

Utility

- For **passive reinforcement learning**:
  - **Definition:** The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed.
    $$U^\pi(s) = E\big[\sum_{t=0}^{\infty} \gamma^t R(S_t)\big]$$
  - **Bellman equations:** The utility values obey the Bellman equations **for a fixed policy**
    $$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s,\pi(s))U^\pi(s')$$
  - **ADP:**Beyes Reinforcement Learning:
    $$\pi^* = \arg\max_\pi \sum_h P(h|e)u_h^\pi$$
    Robust Control Theory:
    $$\pi^* = \arg\max_\pi min_h u_h^\pi$$
  - **Temporal-difference learning:** When a transition occurs from state s to state s', we apply the following update to $U^\pi(s)$, α is the learning rate parameter:
    $$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$
- For **active reinforcement learning:**
  - **Bellman equations:**
    $$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s,a)U(s')$$
  - **Q Learning:** The update equation for TD Q-learning is:
    $$U(s) = \max_a Q(s,a)$$
    $$Q(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$
    $$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))f_i$$
  - **Delta Rule:**
    $$\theta_0 \leftarrow \theta_0 + \alpha(u_j(s) - \hat{U}_\theta(s))$$