Search Problems

agent - entity that perceives its environment and acts upon it
state - configuration of the agent and its environment
actions - choices that can be made in a state
Actions(s) - returns set of actions that can be executed in state s
transition model - a description of what state results from performing any applicable action in any state
Result(s, a) - returns the state resulting from performing action a in state s
state space - the set of all reachable states from the initial state by any sequence of actions
goal test - way to determine whether a given state is a goal state
path cost - numerical cost associated with a given path
solution - a sequence of actions that leads from the initial state to a goal state
optimal solution - a solution that has the lowest path cost among all solutions

Node - data structure that stores:
- → state
- → parent node
- → action (to come from parent)
- → path cost (from initial state)

frontier: [initial state]
explored: [ ]

while {
- frontier $\emptyset$ -> $\nexists$ solution
- node ← frontier
- goal state ∈ node -> return solution
- explored ← node
- frontier ← expand node | node ∉ explored, frontier

stack (LIFO) frontier = DFS
search algorithm that always expands the deepest node in the frontier
queue (FIFO) frontier = BFS
search algorithm that always expands the shallowest node in the frontier

Basic Graph Algorithms

n-nodes, m-edges

nodes - array $O(n)$ memory, $O(1)$ retrieval

edges: Adjacency Matrix / Adjacency List

Adjacency Matrix	Adjacency List	
$O(1)$ retrieval	$O(n)$ asymp retrieval	$\rightarrow$ linked list (overhead)
$\theta(n^2)$ memory	$\theta(n+m)$ memory	array of vectors (slow)
dense graphs	sparse graphs	arrays (known # edges)

Graphs

- → Tree - connected acyclic graph with n-1 edges
- → DAG - directed acyclic graph
- → Bipartite - $S+T$ = nodes s.t. $\forall$ edges $\in \{(u \in T, v \in S), (u \in S, v \in T)\}$

Topological Sort

start from node with no incoming edge $O(n^2+m)$

degrees + queue $\theta(n+m)$

Eulerian Circuit

undirected G, find path through every node, trace back

$\exists$ solution iff G-connected, $\forall u$, $deg(u) \% 2 = 0$

Eulerian path, $\exists$ solution iff G-connected, 0 or 2 nodes with odd degree

Hamiltonian path/cycle - visit every node exactly once

MST

undirected weighted graph $G=(V,E)$

$\exists S \subset E$ s.t. minimal total weight, connects all nodes into a tree

Kruskal - check edges with smallest weight $O(m \log m)$

Prim - check adjacent edges with smallest weight $O(n^2+m) / O(m \log n) / O(m + n \log n)$

Search Algorithms

completeness - strategy is guaranteed to find solution given infinite resources
optimality - strategy is guaranteed to find to find the lowest cost path to goal
branching factor b, maximum depth m, shallowest solution depth s

Uninformed Search

DFS - not complete (cycles), not optimal, $O(b^m)$ time, $O(bm)$ space
BFS - complete, optimal (if unweighted), $O(b^s)$ time, $O(b^s)$ space
UCS (Uniform Cost Search) - always explore the lowest (backward) path cost node $\begin{cases} C^* \text{ - optimal path cost} \\ \varepsilon \text{ - minimal cost} \end{cases}$
↳ complete, optimal (if nonnegative costs), $O(b^{C^*/\varepsilon})$ time and space

Informed Search

Greedy Search - always explore the lowest heuristic value node (forward cost)
↳ not complete, not optimal
A^* Search - always explore the lowest total cost node (heuristic + path)
↳ complete, optimal

$g(n)$ - backward (path) cost, used by UCS
$h(n)$ - estimated forward (heuristic) cost, used by greedy search
$f(n) = g(n) + h(n)$ - estimated total cost, used by A^* search
$h(n) = 1 - g(n)$, A^* becomes BFS

admissability - heuristic value is neither negative nor an overestimate
$\boxed{\text{Th}}$ h satisfies admissability constraint $\Rightarrow$ A^* tree search is complete and optimal.
Proof
A - optimal solution, B - suboptimal solution, n - ancestor of A, in frontier
1) $g(A) < g(B)$, since A is optimal
2) $h(A) = h(B) = 0$, $h^*(x) = 0$, x - goal state, $0 \le h(x) \le h^*(x) \Rightarrow h(x) = 0$
3) $f(n) \le f(A)$, $f(n) = g(n) + h(n) \le g(n) + h^*(n) = g(A) = f(A)$, $h(A) = 0$
1)+2) $\Rightarrow f(A) = g(A) + h(A) = g(A) < g(B) = g(B) + h(B) = f(B) \Rightarrow f(A) < f(B)$
+3) $\Rightarrow f(n) \le f(A) \wedge f(A) < f(B) \Rightarrow f(n) < f(B) \Rightarrow n$ is expanded before B.

consistency - heuristic underestimates the cost/weight of each edge

Th| h satisfies consistency constraint -> A^* graph search is complete and optimal.

Proof

n' - successor of n

$\Rightarrow f(n') = g(n') + h(n') = g(n) + cost(n, n') + h(n') \geq g(n) + h(n) = f(n)$

$\Rightarrow$ If node is removed for expansion, its optimal path has been found.

$h(A) = h(B) = 0 \Rightarrow f(A) = g(A) < g(B) = f(B) \Rightarrow$ Optimal solution found before suboptimal.

consistency -> admissability

dominance, $\forall n,\ h_a(n) \geq h_b(n)$, a is dominant over b

trivial heuristic, $h(n) = 0$, A^* becomes UCS

Adversarial Search

actions - deterministic or stochastic (probabilistic)

zero-sum - agent gain is directly equivalent to opponent's loss

deterministic zero-sum games - Pacman, Checkers (both solved), Chess

adversarial search -> strategy/policy

Minimax

Assumption: Opponent is optimal, will perform worst for us move.

state value - optimal score by agent which controls that state

terminal state - game ends

$\forall$ non-terminal states, $V(s) = \max_{s' \in successors(s)} V(s')$, $\forall$ terminal state, $V(s)$ = known

Game tree - two agents switch off on layers they "control"

maximize utility over children of nodes controlled by the agent, minimize over opponent's

$\forall$ agent-controlled states, $V(s) = \max_{s' \in successors(s)} V(s')$ | $\forall$ terminal states, $V(s)$ = known

$\forall$ opponent-controlled states, $V(s) = \min_{s' \in successors(s)} V(s')$ | DFS or postorder traversal, $O(b^m)$ time

Alpha-Beta Pruning

Stop evaluating children when n's value can at best equal the optimal value of parent.

$O(b^{m/2})$ time

Evaluation Functions

- estimate the true minimax value of node, given the state

depth-limited minimax - non-terminal nodes at max depth treated as terminal

evaluation function - linear combination of features, $Eval(s) = \sum_{i=0}^{n} w_n f_n(s)$

Expectimax

introduction of chance nodes → consider average case, expected utility/value

$\forall$ agent-controlled states, $V(s) = \max_{s' \in successors(s)} V(s')$ | $\forall$ terminal states, $V(s)$ = known

$\forall$ chance states, $V(s) = \sum_{s' \in successors(s)} p(s'|s) V(s')$

Utilities

principle of maximum utility - rational agents maximize expected utility

$A \succ B$ - A is preferred over B, $A \sim B$ - indifferent

Axioms of Rationality:

- → Orderability: $(A \succ B) \vee (B \succ A) \vee (A \sim B)$
- → Transitivity: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
- → 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: $A \succ B \Rightarrow \quad p \geq q \Leftrightarrow [p, A; (1-p), B] \succ [q, A; (1-q), B]$

Minimax

S_0: initial state

Player(s): returns which player to move in state s

Actions(s): returns legal moves in state s

Result(s,a): returns state after action a taken in state s

Terminal(s): checks if state s is a terminal state

Utility(s): final numerical value for terminal state s

function Max-/Min-Value(s):

    if Terminal(s) → return Utility(s)
    v = -/+ ∞, for a in Actions(s): v = Max/Min(v, Min-/Max-Value(Result(s,a)))
    return v

Monte Carlo Search Tree (MCST)

1) Tree traversal

$$UCB1(s_i) = \bar{V}_i + c\sqrt{\frac{\ln N}{n_i}}, \quad C=2$$

2) Node expansion
3) Rollout (random simulation)
4) Backpropagation