Lecture 1 - Introduction to Heuristic Search

agent - entity that perceives its environment and acts upon it
state - a configuration of the agent and its environment
initial state - the state in which the agent begins
actions - choices that can be made in a state
f: Actions (s): return the 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
f: Result (s,a): return the state resulting from performing action a in state s
state space - the set of all states reachable 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

DFS - always explore deepest node in frontier (stack = FILO)
- -> memory efficient
- -> stuck in deep solutions
- -> worst-case: visit all nodes

BFS - always explore shallowest node in frontier (queue = FIFO)
- -> finds shortest path (if unweighted)
- -> memory intensive
- -> worst-case: visit all nodes

Iterative Deepening
- depth-bounded DFS

Informed Search:
(Greedy) Best-First Search
- -> always explore best node in frontier
- -> measure by heuristic h(u)
- -> priority queue

Algorithm	Complete? finds soln if one exists	Time	Space	Optimal? finds shortest path
DFS	if $D < \infty$	$O(b^D)$	$O(b \cdot D)$	–
BFS	✓	$O(b^d)$	$O(b^d)$	if unweighted
IterDFS	✓	*exercise*	*exercise*	–
BestFS	if $D < \infty$	$O(b^D)$	$O(b \cdot D)$	–
Dijkstra	✓	$O(b^e)$	$O(b^e)$	✓
A*	if $D < \infty$	$O(b^e)$	$O(b^e)$	✓

b: (max) branching factor of tree D: depth of tree d: depth of solution
e: effective depth, depends on heuristic

Lecture 2 - A* Search

Informed Search - heuristic function estimates forward cost

Dijkstra = Uniform Cost Search

- -> cost-from-start-to-current-node => guaranteed shortest path

A* Search - look back (path cost) and forward (heuristic estimate)

- -> Disadvantage: entire explored region in memory

Heuristic Properties

- -> admissable - never over-estimate the true cost
- -> consistent - never over-estimate the growth of the path cost

admissable: $0 \le h(n) \le h^*(n)$, consistent: $h(n) \le h(n') + c(n, n')$, consistent -> admissable

Optimality - finds the shortest path

- -> Tree Search: admissable heuristic -> A* - optimal, $h = 0 \rightarrow A^* \equiv UCS$
- -> Graph Search: consistent -> A* optimal, UCS - optimal

Optimally efficient - does least work for any such algorithm

consistent heuristic -> A* - optimally efficient

Efficiency - set od nodes expanded (not # expansions)

Dominance: $\forall n: h_1(n) \ge h_2(n) \rightarrow h_1 \ge h_2$

- -> larger is better (if still admissible)
- -> trivial heuristic is worst (admissible)
- -> exact heuristic is best, but expensive

Non-dominating heuristics: $h'(n) = \max(h_1(n), h_2(n))$ - admissible and dominant

Algorithm	Complete? finds soln if one exists	Time	Space	Optimal? finds shortest path
DFS	if $D < \infty$	$O(b^D)$	$O(b \cdot D)$	–
BFS	✓	$O(b^d)$	$O(b^d)$	if unweighted
IDDFS	✓	*exercise*	*exercise*	–
BestFS	if $D < \infty$	$O(b^D)$	$O(b \cdot D)$	–
Dijkstra	✓	$O(b^e)$	$O(b^e)$	✓
A*	if $D < \infty$	$O(b^e)$	$O(b^e)$	if h admissible
IDA*	if $D < \infty$	$O(b^e)$	$O(b \cdot D)$	if h admissible

b: (max) branching factor of tree D: depth of tree d: depth of solution
e: effective depth, depends on heuristic

Lecture 3 - Adversarial Search

Terminology of Games

single / <u>two</u> / multi-player
Solitaire / Chess / Catan

simultaneous / <u>sequential</u> moves
Rock-paper-scissors / Go

stochastic / <u>deterministic</u> moves
Poker / Chess

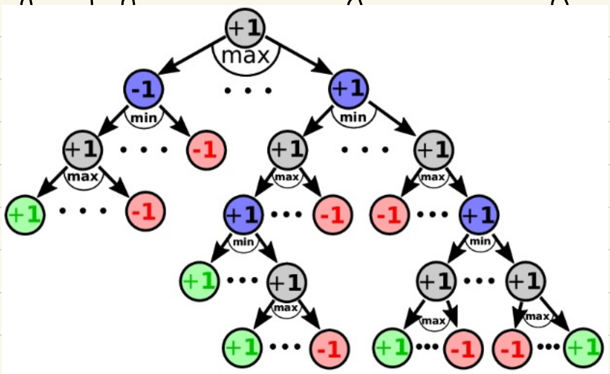
partial / <u>perfect</u> information
Card-games / Connect-four

<u>discrete</u> / continual time
turn-based / real-time strategy

<u>zero-sum</u> / general-sum
Chess / Catan

Minimax Search

Agent plays value-maximizing moves, assuming adversary plays value-minimizing ones.



Improvements:
- depth-limited
- prunning
- stochastic search

evaluation function:
- efficient neural network (NNUE)

Alpha-Beta Prunning

```
fun minimax(n: node, d: int, min: int, max: int): int =
  if leaf(n) or d=0 return evaluate(n)
  if n is a max node
     v := min
     for each child of n
        v' := minimax (child,d-1,v,max)
        if v' > v, v:= v'
        if v > max return max
     return v
  if n is a min node
     v := max
     for each child of n
        v' := minimax (child,d-1,min,v)
        if v' < v, v:= v'
        if v < min return min
     return v
```

Lecture 4 - Monte Carlo Search Tree

Markov Decision Process

Deterministic episodic two-player MDP $\langle S, \mathcal{A}, \mathcal{P}, \mathcal{T}, \mathcal{R} \rangle$

- -> $s \in S$ - all game-states [discrete, perfect information]
- -> $a \in \mathcal{A}(s)$ - all actions that can be played in state s [discrete, sequential]
- -> $\mathcal{P}(s, a) = s'$ - determines next state $s' \in S$ [deterministic moves]
- -> $\mathcal{T}(s) \in \{\perp, \top\}$ - whether game terminates in s [episodic]
- -> $\mathcal{R}(s) \in \mathbb{R}$ - reward for terminal state s
- -> $p(s) \in \{-1, 1\}$ - which player's move it is [two sequential players]

! $\mathcal{R}(s) = +1$ for winning, 0 for draw, -1 for loss

Zero-sum - "one players' gain is the other's loss"

- every reward r for one player is a punishment -r for the other
- two-player zero sum game can be expressed by 1 function
- player 1 maximizes function, player -1 minimizes function

Random Search

Approximate $V(\mathcal{P}(s,a)), \forall a \in \mathcal{A}(s)$ with random rollouts

$$\tilde{V}_R(s') = \frac{1}{N(s')} \sum_{t=1}^{N(s')} r_t, \quad \forall s' \in \{\mathcal{P}(s,a) \mid a \in \mathcal{A}(s)\}$$

minimax selection: $a_t := \arg\max_{a \in \mathcal{A}(s_t)} p(s_t)\, \tilde{V}_R(\mathcal{P}(s_t, a))$

$$\lim_{N(s) \to \infty} \tilde{V}_R(s) = \begin{cases} \mathcal{R}_s & \text{if } \mathcal{T}(s) \\ \frac{1}{\mathcal{A}_s} \sum_{a \in \mathcal{A}(s)} \tilde{V}_R(\mathcal{P}(s,a)) & \text{otherwise} \end{cases}$$

Monte Carlo Tree Search

partial tree $B \subseteq S$

- -> minmax-selection in partial tree
- -> expand tree with new leaf
- -> approximate leaf values with random rollouts
- -> backup resulting reward to all parents

$\mu(a|s)$ - fraction of action a chosen in state s

UCT = MCTS + UCB

exploitation = minimax-like

exploration = random

$$\arg\max_{s'} \left(\frac{w_i}{n_i} + c\sqrt{(\ln N_i)/n_i} \right)$$

w_i - # wins for s' after i^{th} move

n_i - # simulations for s' after i^{th} move

N_i - # simulations for s (parent of s') after i^{th} move

c - exploration parameter, $\sqrt{2} \to$ minimax

$$\tilde{V}_\mu(s) = \begin{cases} \mathcal{R}(s) & , \mathcal{T}(s) \\ \sum_{a \in \mathcal{A}(s)} \mu(a|s)\, \tilde{V}_\mu(\mathcal{P}(s,a)) & , s \in B \\ r_t & , \text{otherwise} \end{cases}$$