Search Methods

Tree Search	Expand nodes using gradients
Graph Search	Avoids revisiting nodes and ensure efficiency

Uninformed search

Uniform cost search	aka Cheapest-first Add visited node to Explored and add its neighbors to the frontier Visit cheapest node in the frontier Move to next cheapest if all neighbors are explored
Iterative deepening	Iteratively calls depth-limited search Initialize frontier with root If not goal, remove from frontier and expand If at the end of depth, terminate and start over Repeat until goal is found Guaranteed to find optimal path More efficient than DFS Time complexity: O(b^d) Space complexity: O(d)
Bidirectional	Finds shortest path of a grpah

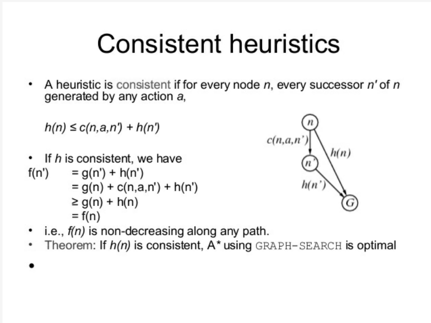
Informed Search

Best-first search	Choose unvisited node with the best heuristic value to visit next Same as lowest cost BFS
Greedy best-first search	Uses heuristic to get the node closest to the goal Bad performance if heuristic is bad Does NOT consider cost of getting to the node

Informed Search (cont)

A* search	Always expand to node with minimum f Evaluate cost of getting to goal using heuristics f(n) = g(n)+h(n) where g is cost to get to n Uses priority queue
Heuristics	Cost to get to the goal
Admissible herustic	Optimistic model for estimating cost to reach the goal Never overestimates h(n) <= c(n) where c is actual cost
Consistent heuristic	h(n) <= c(n, a, n') + h(n') Immediate path costs less than longer path Consistent ⟹ Admissible

Consistent heuristic



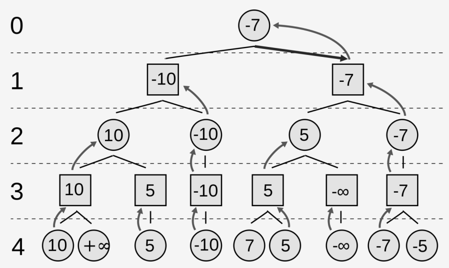
Adversarial Search

Hill climbing	Method of local search Only move to neighbors to find the max Does NOT guarantee to find optimal
Simulated annealing	Method of local search Combine hill climbing and random walk Always find global max

Adversarial Search (cont)

Local beam	Generate k random states Generate successors of all k states If goal stop; else, pick k best successors and repeat Different from hill-climbing since information is shared between k points
Genetic algorithm	Cross-Over and mutation Decomposes strands of DNA and permute Produces children by: Selection, Cross-over, Mutation

Minimax Tree



◎ Max node
☐ Min node

α-β Pruning

```
function alphabeta(node, depth, α, β,
maximizingPlayer)
  if depth = or node is a terminal node
      return the heuristic value of node
  if maximizingPlayer
      v := -∞
      for each child of node
          v := max(v, alphabeta(child, depth - 1,
α, β, FALSE))
          α := max(α, v)
          if β ≤ α
              break ( β cut-off )
      return v
  else
      v := ∞
      for each child of node
          v := min(v, alphabeta(child, depth - 1,
α, β, TRUE))
          β := min(β, v)
          if β ≤ α
              break ( α cut-off )
      return v
```

SAT

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q	¬P V Q
False	False	True	False	False	True	True	True
False	True	True	False	True	True	False	True
True	False	False	False	True	False	False	False
True	True	False	True	True	True	True	True

Propositional SAT: Graph coloring

At lest 1 of K per i	(Ci,1 ∨ Ci,2 ∨ ... ∨ Ci,k) O(n) clauses
1 ≥ color per i	∀ k≠k' (¬Ci,k ∨ ¬Ci,k') O(n^2)
If node i and j share an edge assign different colors	∀ x∈k, (¬Ci,x ∨ ¬Cj, x)