Reflex Agents: choose based on cur percept; PlannG Agent: on consequence of act

search Prob: ① state space ② successor func ③ start & goal

①'s size: #world states = (Agent pos) × 2^m × n^a × b #eat all dot State = Agent pos × 2^m

m: # food n: Ghost pos a: #ghost b: #direction # path state = Agent pos

Tree Search: Depth-First Search b叉树, 深度m, $O(b^m)$ time $O(bm)$ space (stack) × optimal

Breadth-First Search (queue) solution深度s (≤m) $O(b^s)$ time $O(b^s)$ space optimal if cost are all 1

DFS+BFS: run DFS with depth limit = 1, 2, 3 ... Cost-Sensitive: Uniform Cost Search (optimal).

UCS priority Queue solution cost C, arc cost ≥ ε, 深度s约 = C/ε, $O(b^{C/\varepsilon})$ time & space

Heuristic: h(goal) Greedy Search: min[h(goal)] A* Search: UCS + Greedy $f(n) = g(n) + h(n)$

$0 \le h(n) \le$ true cost($h^*(n)$): Admissible. $h(A) - h(C) \le$ cost(A to C) optimal if h admissible

---

CSP (Constraint Satisfaction Problems) goal test = set of constraints

回溯S: ① 1 var/time ② do × conflict prev for DFS FilterG: detect fail earlier

Forward CheckG: backtrack when any var has × value left strong k-consisT also (k-1) ··· 2-con..

$O(n^2d^3)$ time, can be $O(n^2d^2)$ n: # color d: # place

[k-consistency: For each k nodes, any consistent assignM to (k-1) can be extended to the kth node] 取值顺序 (ordering):

① MRV: Minimum RemainG Values (最小剩余值)

② LCV: Least ConstrainG Value (最小约束值): rule out fewest val in remain vars

graph of n var → subG of c var → worst case: $O((n/c)(d^c))$ (n: # vals)

CSP $O(nd^2)$ time if constraint Graph has no loop. (normally $O(d^n)$)

Tree CSP: Remove backward: for i = n : 2 remove Inconsistent($P(X_i)$, X_i)

$O(nd^2)$ Assign forward: for i = 1 : n, assign X_i consistently with Parent(X_i)

cutset: remain constraint G is a tree. Initiate Cutset → solve residual

Iterative Algorithm: while × solve, randomly pick conflict var & assign val with min constraints

Local Search: hill climbing → repeat until no better neighbour

Beam Search: keep k states in 爬山法 模拟退火: 允许 dowhill moves

accept downhill with $P = e^{-\Delta E/T}$, time T↑, P↓

Adversial S (对抗搜索) Minimax: deterministic, zero-sum games

each node's minimax value: best achievable utility against a rational (optimal) adversary $O(b^m)$ time $O(bm)$ space 分支因子 (branchG fac) limit by Consider Only Good Moves

Evaluation Functions: Eval(s) = weight * factors (features)

剪枝 α-β pruning $O(b^{m/2})$ time with perfect Ordering

Expectimax search: compute the average score under optimal play

Environment is an extra "random agent" player that moves after each min/max agent

---

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (Xi, Xj) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue

function REMOVE-INCONSISTENT-VALUES(Xi, Xj) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```

- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$

Knowledge base: set of sentences in a formal language to represent knowledge about the "world" Inference engine: answers any answerable question following the knowledge base

Syntax: whether is a Sentense. Conjunction: ∧

Semantic: T/F in a World disjunction: ∨ Imply: → biconditional ↔

(→) is F only when (T→F) Valid: always true Satisfiable: can be True. unSatisfiable: always F

Logical Equivalence

$(\alpha \land \beta) \equiv (\beta \land \alpha)$ commutativity of ∧

$(\alpha \lor \beta) \equiv (\beta \lor \alpha)$ commutativity of ∨

$((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))$ associativity of ∧

$((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))$ associativity of ∨

$\neg(\neg\alpha) \equiv \alpha$ double-negation elimination

$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition

$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \lor \beta)$ implication elimination

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$ biconditional elimination

$\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)$ de Morgan

$\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)$ de Morgan

$(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))$ distributivity of ∧ over ∨

$(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))$ distributivity of ∨ over ∧

α entails β: $\alpha \models \beta$ (α)β Axiom: a sentence known to be true

Inference: Sound & Complete everyth can be Proved ⇔ is entailed

Conjunctive Normal Form: (∨)∧(∨)∧(∨) Horn logic (Single form):

$P_1 \land P_2 \land P_3 \cdots \land P_n \Rightarrow Q$ OR $\neg P_1 \lor \neg P_2 \cdots \lor \neg P_n \lor Q$ All are Universal

Resolution Prob (Prove by ContradicT); for/backward chainG

First-Order logic ① Objects; ② Relations; ③ Functions Logical Sym includes ∧∨; ∀∃; x y; = connectives. Non-logical Sym: Constant; Predicte (Bro, >); Func (Sqrt...)

Atomic Sentence = predict(term1, ... tn) or t1 = t2; Term = const, or var or func(t1...tn)

↳ + Connectives → Complex Sentence var is bound if assigned ∀ OR ∃ is free if w/o ∀ or ∃

use ∀ with ⇒, ∃ with ∧

- ∀x ∀y is the same as ∀y ∀x Property of quantifiers
- ∃x ∃y is the same as ∃y ∃x
- ∃x ∀y is not the same as ∀y ∃x
  - ∃x ∀y Loves(x,y)
    "There is a person who loves everyone in the world"
  - ∀y ∃x Loves(x,y)
    "Everyone in the world is loved by at least one person"
- Quantifier duality: each can be expressed using the other
  ∀x Likes(x,IceCream) ≡ ¬∃x ¬Likes(x,IceCream)
  ∃x Likes(x,Broccoli) ≡ ¬∀x ¬Likes(x,Broccoli)

every var must be bound in FOL

Universal instantiation (UI) with ∀, can change x into any sentence Existential I (EI) ∃x; change x to C_1 (new const Sym) <skolem constant>

Resolution: ① (A∨B)∧(¬B∨C) 得 A∨C sound & complete

② A⇒B 和 A 得B ③ ¬A∨B 和 A 得B ¬∀x, P ≡ ∃x ¬P

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```