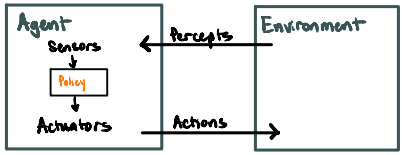
1 Introduction

- An agent perceives its environment through sensors and acts upon it through actuators
- A rational agent chooses action to max expected value



2 Search

Search Problems have:

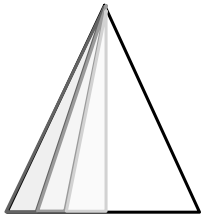
- State Space S
- Initial State s_0
- Actions in state $A(s)$
- Transition model $Result(s,a)$
- Goal test $G(s)$
- Action cost $c(s,a,s')$

Solutions: action sequence that reaches goal state

Optimal Solution: least cost among solutions

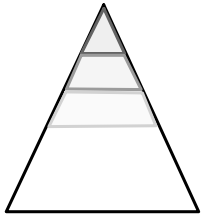
Depth First Search (DFS)

Strategy: expand a deepest node first
Use: LIFO stack
Optimal?: No, finds "leftmost" solution



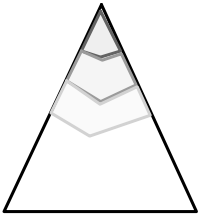
Breadth First Search (BFS)

Strategy: expand shallowest node first
Use: FIFO Queue
Optimal: If costs are equal



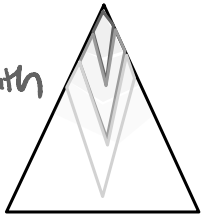
Uniform Cost Search (UCS)

Strategy: expand lowest cost from root $g(n)$
Use: Priority Queue sorted by $g(n)$
Optimal?: Yes



A* Search

Strategy: expand node most likely to be on optimal path
Use: Priority Queue sorted by $g(n) + h(n)$
Optimal: Yes, if admissible and consistent



Admissible heuristic: $0 \leq h(n) \leq h^*(n)$ ← true cost to goal

heuristic ≤ actual cost

Consistent heuristic: $h(A) - h(C) \leq c(A,C)$

heuristic "arc" cost ≤ actual cost for each arc

Greedy Search

Strategy: selects frontier node w/ lowest heuristic value for expansion, believes closest to goal

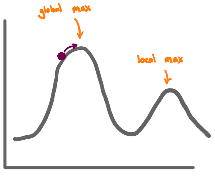
Use: priority queue with heuristics, estimated forward cost

Optimal: No, not guarenteed to find goal state, unpredictable

Local Search

Hill Climbing

Idea: Start wherever, repeat: move to the best neighboring state
If no neighbors better than current, quit




```
function HILL-CLIMBING(problem) returns state
    current ← make-node(problem.initial state)
    loop do
        neighbor ← highest valued successor of current
        if neighbor.value ≤ current.value then
            return current.state
        current ← neighbor
```

Random-restart hill climbing trivially complete: restarts from random initial state

Simulated Annealing

Idea: random walk + hill climbing, choose randomly, choose worse w/ some probability according to temperature, temp starts high w/ more "bad" moves allowed and decreases

function SIMULATED_ANNEALING (problem, schedule) returns state

current ← problem.initial-state

for t←1 to ∞ do

T ← schedule(t)

if T = 0 then return current

next ← a randomly selected successor of current

ΔE ← next.value − current.value

if ΔE > 0 then current ← next

else current ← next w/ prob $e^{\Delta E/T}$

If T decreased slowly enough, will converge to optimal state

Local Beam Search

multiple interaction searches

Idea: track k states at each iteration, k threads share information good threads attract other threads, chooses k best successors

Genetic Algorithms

Break and recombine states

Idea: beam search k states in population, each state evaluated w/ evaluation function (fitness func), offspring produced by crossing parent strings at crossover point

3 Games

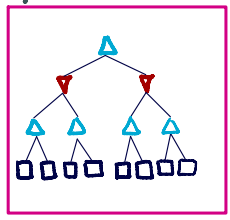
Zero sum games: our gain is directly equivalent to opponent's loss

Minimax

algorithm to counter optimal opponent moves

Idea: zero sum game algorithm assuming opponent plays optimally

terminal utilities/state value: optimal score attainable



function minimax-decision(s) returns action
return action a in Actions(s) w/ highest minimax_value(Result(s,a))

function minimax_value(s) returns a value
if Terminal-Test(s) then return Utility(s)
if Player(s) = MAX then return $\max_{a \in Actions(s)}$ minimax_value(Result(s,a))
if Player(s) = MIN then return $\min_{a \in Actions(s)}$ minimax_value(Result(s,a))

Alpha-Beta Pruning

optimization to minimax: $O(b^m) \Rightarrow O(b^{m/2})$ (m: tree depth, b: branching factor)

Idea: Stop looking as soon as you know n's value can at best equal optimal value of n's parent

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
```

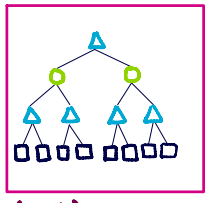
Evaluation functions are used in depth-limited minimax to output an estimate of the true minimax value of the node

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

weight associated w/ feature — features of state

Expectimax

algorithm with randomness using Expected value

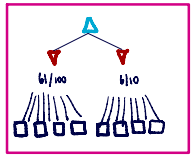


Idea: Introduce chance nodes calculated by taking expected utility of children

function decision(s) returns action
  return action a in Actions(s) w/ highest value(Result(s,a))

function value(s) returns a value
  if Terminal-Test(s) then return Utility(s)
  if Player(s) = MAX then return $\max_{a \in Actions(s)}$ value(Result(s,a))
  if Player(s) = MIN then return $\min_{a \in Actions(s)}$ value(Result(s,a))
  if Player(s) = CHANCE then return $\sum_{a \in Actions(s)}$ Pr(a) * value(Result(s,a))

Monte Carlo Tree Search (MCTS)



Idea:

1) Evaluation by rollouts: From state s play many times using policy and count wins and losses

2) Selective search: explore parts of the tree, without constraints on horizon, that will improve decision at root

UCB Algorithm

total wins for Player(Parent(n))

$$UCB1(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{\log N(PARENT(n))}{N(n)}}$$

User specific param to balance exploration vs. exploitation

total rollouts from n

4 Logic

Maintain a knowledge base w/ logical sentences that can make logical inferences

Logic Syntax

Symbol	Meaning	Description
$\neg$	not	$\neg P$ true iff P is false
$\wedge$	and (conjunction)	$A \wedge B$ true iff both A true and B true
$\vee$	or (disjunction)	$A \vee B$ true iff either A true or B true
$\Rightarrow$	implication	$A \Rightarrow B$ true unless A true and B false
$\Leftrightarrow$, $\equiv$	biconditional	$A \Leftrightarrow B$ ($A \equiv B$) true iff either both A,B true or false

Conjunctive normal form (CNF): conjunction of clauses which are disjunction of literals

$$(P_1 \vee \dots \vee P_i) \wedge \dots \wedge (P_j \vee \dots \vee P_n)$$

Converting to CNF:

1) Eliminate $\Leftrightarrow$
2) Eliminate $\Rightarrow$
3) Not's ($\neg$) only on literals
4) Reduce and distribute

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$$

$$\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta), \quad \neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$$

model: assignment of true/false to all proposition symbols
valid: true in all models
satisfiable: at least one model where true
unsatisfiable: not true in any models

First Order Logic (FOL)

uses objects as base components

Use function symbols to name objects in terms

Operator Precedence
$\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

quantifiers:

- universal quantifier $\forall$: "for all"
- existential quantifier $\exists$: "there exists"

Propositional Logical Inference

Entails ($\models$): in all models where A is true, B is as well, $M(A) \subseteq M(B)$

Show by:

1) $A \models B$ iff $A \Rightarrow B$ valid — direct proof
2) $A \models B$ iff $A \wedge \neg B$ unsatisfiable — proof by contradiction

Model Checking

DPLL Algorithm

depth-first, backtracking search w/ tricks SAT solver

Idea: Solve satisfiability problem given in CNF, continue assigning truth values until found or cannot be found, backtrack w/ conditions:

1) Early Termination: clause true if any symbol true, sentence false if any single clause is false
2) Pure Symbol Heuristic: symbol only shows up in pos/neg form assigned immediately $(A \vee B)(\neg B \vee C) \wedge (\neg C \vee A)$ A=True
3) Unit Clause Heuristic: Clause w/ one literal or literal w/ many falses

Theorem Proving

Resolution Algorithm

Idea: iteratively applies it to knowledge base either q inferred or nothing left to infer

Forward Chaining Algorithm

Idea: data driven reasoning, iterating through every implication statement where LHS known, adding RHS to facts using Generalized Modus Ponens

Solve inference in FOL by Generalized Modus Ponens or propositionalization, translating problem into propositional logic and using SAT solver

5 Probability

Marginal distribution: $P(A,B) = \sum_C P(A,B,C)$

Bayes' rule: $P(A|B,C) = \frac{P(A,B,C)}{P(B,C)} = \frac{P(A,B|C)}{P(B|C)}$

Chain Rule: $P(A,B,C) = P(A,B|C)P(C) = P(A|B,C)P(B|C)P(C)$

A conditionally independent given C: $P(A,B|C) = P(A|C)P(B|C)$

A independent of B given C: $P(A|B,C) = P(A|C)$

A,B independent: $P(A,B) = P(A)P(B)$

$P(Q_1 \dots Q_k | e_1 \dots e_k)$

Query Variables: unknown, trying to compute

Evidence variables: observed, values known

Hidden Variables: values present but not in distribution we are computing

Bayes Nets

Idea: Joint distribution distributed across smaller probability tables with Directed Acyclic Graph (DAG)

1) DAG w/ node per variable X
2) conditional distribution for each node $P(X|A_1 \dots A_n)$ where A_i is i^{th} parent of X, stored as conditional probability table (CPT)

Each node is conditionally independent of all ancestor nodes in graph, given all of parents

Bayes Net Inference (calculating joint probability)

Eliminate variables by:

1. Join (multiply together) all factors involving X
2. Sum out X

Prior Sampling

Randomly generate samples

Rejection Sampling

early reject any sample inconsistent w/ evidence

Likelihood Weighting

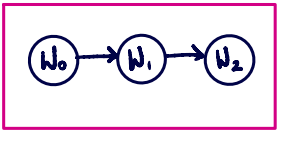
Ensure we never generate a bad sample

1) manually set variables to equal evidence
2) Weight each sample by probability of the evidence variables given the sampled variables

Gibbs Sampling

Set all variables to random value, repeatedly pick one variable at a time, clear value, resample

6 Markov Models



Chain like infinite-length Bayes Net

Need to know initial state and transition model

Mini-Forward Algorithm

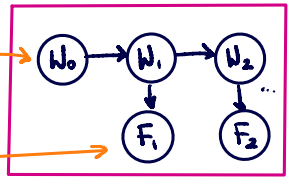
$$Pr(W_{i+1}) = \sum_{W_i} Pr(W_{i+1} | W_i) \, Pr(W_i)$$

Stationary Distribution

$$Pr(W_{t+1}) = Pr(W_t) = \sum_{W_t} Pr(W_{t+1} | W_t) \, Pr(W_t)$$

Hidden Markov Models

allows to observe some evidence at timestep and affects belief distribution



Forward Algorithm

$$B'(W_{t+1}) = \sum_{W_i} Pr(W_{t+1} | W_i) B(W_i)$$ Time elapse update

$$B(W_{t+1}) \propto Pr(f_{t+1} | W_{t+1}) B'(W_{t+1})$$ Observation update

$$B(W_{t+1}) \propto Pr(f_{t+1} | W_{t+1}) \sum_{W_i} Pr(W_{t+1} | W_i) B(W_i)$$

Viterbi Algorithm

Dynamic Programming algorithm to solve

$$\underset{X_{1:N}}{\text{argmax}}\, P(X_{1:N} | e_{1:N}) = \text{argmax}_{X_{1:N}} P(X_{1:N}, e_{1:N})$$

$$m_t[x_t] = P(e_t | x_t) \, m_{t-1} P(x_t | x_{t-1}) \, m_{t+1}[x_{t-1}]$$