Agents and Rationality

An agent is something that acts
operate autonomously, perceive environment, persist and adapt

Rational agent acts to achieve the best (expected) outcome

Agents perceive environment through sensors and acts upon through actuators

Agent function is abstract mathematical description, agent program is concrete implementation

Performance measure determines what behavior is rational, should design with desired environment in mind

Omniscience is knowing the actual outcome

Agent should learn, autonomy measures reliance on designer prior knowledge

Environments

Task environment consists of performance measure, environment, actuators and sensors

Fully/partially observable - agents sensors give it full/partial knowledge of environment

Single/multiagent - other agent B may try to maximize own performance measure based on A, see if you want to treat B as agent or object

Deterministic/stochastic - if next state is completely determined by current state and agent action, stochastic → probabilities, partially observable may appear to be stochastic, nondeterministic → no probabilities

Episodic/sequential - current decision does not affect later decision (part checking vs. chess/driving)

Static/dynamic - If environment changes while agent deliberates → dynamic (taxi), doesn't change but performance measure does → semi-dynamic (clock), otherwise static (crossword)

Discrete/continuous - time handling (chess → discrete), taxi input from digital source → continuous

Known/unknown - known if outcome/probabilities all known, unknown → do not know how environment responds

Types of Agents

Simple Reflex Agents

Base actions off of current percept, no history

Use percept in lookup table, assumes fully obs

Infinite loops may happen, can randomize actions

Model-based Reflex Agents

Maintain internal state based on percept history

Use model of the world, update state based on percept/action

Creates a "best-guess" of world state

Goal-based agents

Make decision based on state as well as goal

May make some decisions to reach goal specifically, don't crash if you want to make it

Goal-based more flexible than reflex-based

Utility-based agents

Generalization of goal-based agents for optimality

Rational utility-based agent maximizes expected utility

Can make proper tradeoffs, but we must still define

Search Problems

Generally, environment is observable, discrete, known and deterministic

"Formulate, search, execute" - agent carries out solution with eyes closed after searching

Problem (in parts)

1. Initial State - where/when agent begins
2. Actions - description of possible available actions
3. Transition Model - description of what actions do, i.e. Result(s, a) = s'

(1–3 share state space)

Forms graph where nodes are states and links = actions

4. Goal Test - check if s matches goal state
5. Path cost function - assign numeric cost to path

step-cost => c(s, a, s')

Optimal solutions entail lowest path cost

Tree-search + explored set => Graph-search

In each node n in search tree, remember:

- n.State - state in state space
- n.parent - parent
- n.Action - action applied from parent
- n.Path-Cost - path cost so far by parent pointers

Use different types of queues for fringe

Completeness - will the algorithm find a solution if there is one?

Write complexity in terms of branching factor b, depth d and m, max length of path in state space

Uninformed Search Strategies

Breadth-first search (queue as fringe)

Goal Test applied when node generated, vs. expanded

Complete and will find shallowest node d, not always optimal

Time + Space complexity - $O(b^d)$, dominated by size of the frontier

Uniform-cost search (priority queue as fringe)

Expands the node with lowest path cost g(n)

When node n expanded, optimal path to n is known

Complete if cost of every step exceeds small positive ε

Time/space complexity = $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$

Depth-first search (stack as fringe)

Graph search in finite state space complete, tree search is not (infinite loops)

Definitely not optimal by search nature

Time Comp - $O(b^m)$, all nodes

Space - $O(bm)$ - good for memory

Agent Program Representations

Atomic "Black Box"

Each state of the world is indivisible and has no internal structure.

Factored Representation

Split state into set of variables and attributes, each has a value

Can use booleans and continuums

Able to represent uncertainty (leave blank)

Structured Representation

Model how different parts of the world are related to each other

Learning Agents

Learning element - makes improvements

Performance element - makes decisions, fed by learning element

learning elem. uses feedback from critic

Problem generator - suggests to performance element things that will lead to informative experiences

Depth-Limited Search

DFS to predetermined depth limit l

Possible incompleteness if choose $l < d$

Pretends that no children after l

Iterative-deepening DFS $O(b^d)$-time, $O(bd)$-space

Gradually increases depth-limit, expands like BFS

Iterative lengthening search

Increasing path-cost vs. depth-limits

Bidirectional Search

Run BFS/UCS from start and goal, so $b^{d/2} + b^{d/2} < b^d$

Must be able to compute predecessors

Informed (Heuristic) Search

Make use of a combination of path cost g(n) and estimated remaining cost h(n)

Greedy best-first search

f(n) = h(n), tries to expand node closest to goal

Tree-search infinite loop, graph-search is complete

A* Search

f(n) = g(n) + h(n)

Complete + optimal, if h(n) is admissible

tree search is optimal, if consistent, graph search is optimal

consistent if: $h(n) \leq c(n, a, n') + h(n')$

Whenever node expanded, optimal path found because (assuming heuristic is admissible), if n' on way to n, then would've been expanded first

Expands all nodes $f(n) < C^*$, may expand nodes on goal contour

Optimally efficient - no other optimal algorithm guaranteed to expand fewer nodes

Iterative Deepening A*

Bound is f-cost, better from memory

Recursive best-first search

Backtracks to maintain state, does not store

Heuristic Functions

Effective branching factor

$N+1 = 1 + b^* + (b^*)^2 + (b^*)^3 \ldots$

h_2 dominates h_1 => A* never expands more nodes using h_2 than h_1

Relaxed problems can be used to generate admissible + consistent heuristics

Learning + linear combination

$h(n) = c_1 x_1(n) + c_2 x_2(n)$,

not always admissible + consistent

Local Search

Evaluate and modify current state, path cost is not relevant
① use small amounts of memory
② can generally find reasonable solutions in large/infinite spaces
complete - always finds a solution
optimal - always finds global min/max

Hill-climbing search
steepest-ascent/greedy local search
complete-state formulation - all variables are present at one time
ⓐ Local maxima - gets stuck by steepest ascent
ⓑ ridges - sequence of local maxima
ⓒ Plateau - flat area or shoulder
sideways move to escape from shoulder
stochastic hill-climbing - random from uphill moves w/ weighted probability
First-choice hill-climbing - generate random successors until one is better
random-restart hill-climbing - conduct search from random states, sometimes restart

Simulated-annealing
Refine state by picking random move and if better, take it, else accept with probability $\propto T$, which ↓
Shake out of local min/max

Local-beam search
start with k random states
Generate all successors of each state and only keep top k
Parallel search threads, pass info
Stochastic beam-search - choose k successors at random $\propto$ how "good"

Genetic algorithm
similar to stochastic beam search
successors generated by combining two parent states
Objective function as fitness function
crossover point and mutation

Adversarial Search

Definition of game as search problem
S_0: Initial state, how game set up at start
Player(s): which player has move in state
Actions(s): Returns set of legal moves
Result(s,a): Transition model - result of move
Terminal Test(s): check if game over
Utility(s,p): objective/payoff function
Zero-sum => total payoff same in all games
Initial State, Actions, Result define game tree

Minimax
Alternation of maximizing(you) and minimizing (opponent) plies, accomplished by DFS to terminal states
Multiplayer games may use a vector $\langle a,b,c\rangle$ for combined values and may have different pairs
Alpha-beta Pruning
As we search, keep values α(best max value) and β(best min value), cut off exploration if value worse than β or
May use transposition table of known states to cut down
Cutoff search w/ evaluation to make impatient real-time decision
Can modify α-β to use eval beyond a certain point
Good for quiescent positions, horizon effects is further

Searching w/ Nondeterministic Actions

Percepts help in partially observable or nondeterministic environment
contingency plan (strategy) - for all possible sequence of percepts
Solutions look like trees (if-then-else) rather than sequences
And-Or search trees alternate OR(you) and AND(environment), solution must be able to handle at least one OR options and all AND options
Some nondeterministic environments may require cyclical solutions

Searching w/ Partial Observation

Belief state is the set of all possible physical states that an agent could be in.

No Observation
Belief-States: up to 2^N unique sets of belief states
≡ belief-state space, set of possible physical
Initial State: set of all states in P initially
Actions: If illegal actions have no effect, $\bigcup_{s\in b}$ Actions$_P(s)$, else $\bigcap_{s\in b}$ Actions$_P(s)$ => safer
Transition Model: deterministic =>
$b' =$ Result$(b,a) = \{s' : s' =$ Result$_P(s,a)$ and $s \in b\}$
else $\bigcup_{s\in b}$ Results$_P(s,a)$
Goal Test: check that all belief states satisfy
Path cost: May be different between states

With observation
prediction: $\hat{b} =$ Predict(b,a)
Observation prediction: determine set of percepts that could be observed
update: Partitions and returns set of belief states that correspond to input percept o => b_o=update$(\hat{b},o)$
Nondeterminism will just enlarge the number of belief states

Stochastic Games

Introduce chance nodes, value equivalent to expected value of outcomes
minimax => expectiminimax
Alternating a-○-b-○... pattern

Constraint Satisfaction Problems

Utilize a factored representation where variable assignments leads to simultaneous satisfaction of all constraints
X - set of variables $\{X_1,\dots X_n\}$, D - set of domains, one for each var
C - set of constraints => allowable combinations of values
Constraint graphs to visualize systems of binary+ constraints
Global constraint - arbitrary # of variables

node consistency ≡ entire unary, arc - entire binary
path consistency ≡ triples, $\{X_i,X_j\}$ w/ X_m if $\{X_i,X_m\}$, $\{X_m,X_j\}$

Solutions are commutative
Backtracking Search (DFS one var at a time, assign, backtrack if there is no domain), flexible
Heuristics: MRV - minimum remaining value
Degree - prioritize those w/ higher degree
LCV - least constraining value
Forward checking - Choose variable, establish arc consistency from neighbors and continue, backtrack if no domain
MAC (Maintaining Arc consistency) - call AC3 instead of just assigning from neighbors
chronological backtracking + backjumping
min-conflicts (pick random var and minimize its conflicts, repeat)

Cutset conditioning
Choose subset of graph to form a tree
Loop through cutset and run tree CSP solver
Enforces backward consistency then runs forward checking

Logical Agents

Knowledge base ≡ set of sentences in knowledge representation language that describe the world
TELL and ASK operations, both may involve inference, deriving new sentence from old

Logic

entailment implies β follows logically from α
logical inference/model checking
sound - can derive entailed sentences,
complete - can derive any sentence that is entailed

Propositional Logic

$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
truth tables to express truthfulness of sentences
Model-checking, Theorem-proving
Valid - true in all models, satisfiable - true in some model
Prove β from α by unsatisfiability of $\alpha \wedge \neg\beta$, proof by contradiction

Initial state: KB; Actions: All inference rules applied to sentences matching top half of inference rule
result: add sentence in bottom half
Goal: state we try to prove

Conjunctive normal form
DPLL - pure symbol, unit clause → same sign
early termination