CS 188

Fall 2015

8/27

Rationality is defined in terms of achieving maximum utility by some pre-defined metric or set of goals/intentions. Rationality depends only on the usefulness of the choice reached rather than on any aspect of the process that led to that choice. For example, a rational process for playing tic-tac-toe could be formed simply by creating a table for all game states; this would not have consist of any decision process whatsoever.

AI, economics, statistics, operations research, etc. assume utility to be **exogenously specified**. The difficulty of competent machines is a question of value misalignment.

9/1

Sensors are preceptors of the environment and *actuators* are methods by which it manipulates the environment through actions.

An *agent function* maps from percept histories to actions.

$$f : P^* \rightarrow A$$

An *agent program* I runs on some machine M to implement f:

$$f = Agent(I, M)$$

Not all agent functions can be implemented by some agent program. E.g. halting problems, NP-hard problems, chess (combinatorically large amounts of information needed to process)

Use a *performance measure* to evaluate effectiveness. Care must be taken to ensure that the performance measure accurately measures the execution of the task designed. A performance measure is a measure on the *environment*.

A *rational agent* maximizes the expected value of the performance measure. Rationality depends on prior knowledge of environment, action, previous percepts.

PEAS model: performance measure, environment, actuators, sensors.

An *autonomous* agent has behavior determined by its own percepts and experience, with the ability to adapt and learn, without depending solely on built-in knowledge.
Environment characteristics:

observable (fully/partial): the proportion of the environment the agent's sensors give it access to

number of agents

deterministic/stochastic: the degree to which the environment's state may fluctuate, given a particular action (an environment which is deterministic, except for the actions of other agents, is *strategic*)

static/dynamic: the degree to which the environment changes while the agent is deliberating

discrete/continuous, known/unknown (e.g. don't know the dynamics of the system, but still try to maximize utility)

Effects of the environment on agent design:

partially observable $\rightarrow$ agent requires memory

multi-agent $\rightarrow$ randomness may be necessary

static $\rightarrow$ can utilize time to implement a rational decision

continuous time $\rightarrow$ will have some continuously operating controller

Agent types (increasing generality/complexity):

simple reflex

state-based reflex agents

goal-based agents

utility-based agents

Two of these are reflexive, the next two are planning-based.

9/3

Planning agents predict consequences of actions $\rightarrow$ transition model.

Deliberativeness: can generate a complex plan or a simple one and correct rapidly.

A *search problem* consists of a state space, a set of allowable actions, a transition model (corresponding to results), a step cost function, a t_0 and a goal test.

Solution: A_n that transforms a t_0 into a goal state.

A *real world state* is highly general.

A *search state* is specific to the problem, focusing on the vital details of the environment.

Can describe the state space using a directed graph. Note that each state appears only once in the graph. Edges can have costs associated with them.

Search trees incorporate temporal direction.

Implement nodes with state, parent, action, cost of action (path-cost)

Depth-first-search, breadth-first-search, iterative deepening (DFS with limit 1, DFS with limit 2, etc.)

Uniform-cost-search: expand frontier at cheapest node first. A* can prove is optimal. However, uses no information about goal.

Tree searches can often lead to excess/repeated work. Can correct by making a check whether or not potential actions lead to an explored state.

Graph search exists in a finite space, memory $\propto$ runtime.

9/8

Heuristic estimates how close a state is to the goal
Greedy search expands nodes based upon a given heuristic
Uniform-cost search orders by path cost, i.e. *backwards cost*, g(n)
Greedy search orders by proximity to goal, i.e. *forward cost*, h(n)
An A^* algorithm orders by $f(n) = g(n) + h(n)$
Key point: an A^* search must have an effective heuristic to work
An admissible heuristic satisfies $0 \leq h(n) \leq h^*(n)$, with $h^*(n)$ the true cost
A^* search with an admissible heuristic is optimal
A heuristic h_a is dominant over h_c, $h_a \geq h_c$ if $\forall n \; h_a(n) \geq h_c(n)$
A dominant heuristic of an admissible heuristic, if still admissible, is preferable
A maximum over admissible heuristics is a dominant heuristic, and still admissible

9/10

Local search algorithms
Often path can be irrelevant, and the solution is merely the goal stat itself
Search for a configuration satisfying constraints or an optimal configuration
Iterative improvement can be used
Constant space
E.g. n-queens problem: heuristic value, number of conflicts b/t queens

Hill-climbing algorithm
Starts with an arbitrary solution, then incrementally changes a single element
Apply the change if it improves the heuristic
Repeats until no further improvements can be found
Apply random restarts to avoid stalling on local optima
Allow for sideways moves (heuristic does not change) for greater freedom

Simulated annealing
Vary the probability of performing moves of a certain δ based off of thermodynamics
Start at a high temperature, gradually cool
Theorem: simulated annealing finds global optimum with probability 1 given a slow enough cooling schedule

Local beam search
K copies of a local search algorithm
At each iteration, generate all successors from current states
keep best K to be the new current states

Genetic algorithms
Keep best N hypotheses at each step based on a fitness function
Implement pairwise crossover operations, and possibly mutation

Real-world search issues:
non-determinism (now think of solutions as contingency plans)
partial observability (can use a belief state = the state of possible states the agent is in)
in a belief state space, can add an observation model

contingent solution aka branching/conditional plan
    encoded as and-or search trees, where hit goals at the end
and-or search shorthand: or-search the root node
    or-search succeeds if and-search succeeds on outcome set for any action
    and-search succeeds if or-search succeeds on all nodes in outcome set
cyclic solution: every leaf is a goal state, every point in the plan has a path to the leaf
belief state: set of all environment states agent could be in
    more generally, what the agent knows given all percepts to date
formulating a sensorless problem: $Actions_P, Result_P, Goal - Test_P, Step - Cost_P$
    belief state b (set of physical states s)
    goal test: every s in b satisfies $Goal - Test_P(s)$
    actions: componentwise application of $Actions_P$ on each $s \in b$

$$Result(b,a) = \cup_{s \in b} Result_P(s,a)$$

or, if nondeterministic

$$Result(b,a) = \cup_{s \in b} Results_P(s,a)$$

$Step - Cost(b,a,b') = Step - Cost_P(s,a,s')$ for any $s \in b$
improving a sensorless problem
    if any $s \in b$ is unsolvable, b is unsolvable
    If a belief state $b \subset b'$ and b' has a solution, b has the same solution
    Hence if $b \subset b'$ and we have path to b, discard b'
partial observability: belief state transition model
    $b' = Predict(b,a)$ updating the belief state just for the action: this is the same as a transition model for sensorless problems

$$Possible - Percepts(b') = \cup_{s \in b} Percept(s)$$

$$Update(b',p) = \{s \in b' : p = Percept(s)\}$$

(after receiving a percept p)

$$Results(b,a) = \cup_{p \in Possible-Percepts(Predict(b,a))} Update(Predict(b,a),p)$$

percept p given by the environment
$b \leftarrow Update(Predict(b,a),p)$
    This is the predict-update cycle, aka monitoring, filtering, state estimation
    Two special cases: localization and mapping