CS 188, Fall 2015

Intelligent Agents

Rationality: achieving maximum utility by some pre-defined metric or set of goals/intentions.
  depends on the usefulness of the choice and not on the process that led to that choice
  e.g., rational process for playing tic-tac-toe by tabulating game states
  has no decision process, but acts rationally
An agent has sensors which take in percepts and actuators which effect actions
  a percept is a set of perceptual inputs at a fixed point in time
  a percept sequence is composed of percepts
An agent function is $f : P^* \to A$ where P^* is the percept sequence and A the set of actions
  the agent program is the specific architecture which implements this function
  not all agent functions can be implemented by some agent program
  e.g. halting problems, NP-hard problems, "too-large" problems (e.g. chess)
Performance measure: an objective criterion for the success of behavior of an agent
Rational agent: maximizes expected performance measure by its actions
  given the prior knowledge and percept sequence available to it
  e.g. vacuum world: 2 squares, could be dirty or clean
  action function: suck if dirty, move if clean
  under the measure of most clean squares/time period, this is rational
  if we seek to minimize movements as well, this is irrational
Autonomy: the ability to function beyond the prior knowledge of the designer

The task environment: performance measure, environment, actuators, and sensors
  partially vs. fully observable (perceive all aspects relevant to choice of action)
  stochastic vs. deterministic (next state a function only of current state, action)
  strategic: deterministic but for the actions of other agents
  episodic vs. sequential (current decision could affect future decisions)
  static vs. dynamic (environment can change while agent deliberates)
  semidynamic: environment doesn't change with time but performance score does
  semidynamic e.g. chess with a clock
  discrete vs. continuous (can apply to state, time, percepts/actions)
  single vs. multiagent: e.g. competitive multiagent, cooperative multiagent

Agent structure
  the agent program only takes in the current percept
  the agent function maps from the entire percept history

A simple reflex agent uses only the current percept in its decision process
  responds via condition-action rules
  requires full observability to work effectively
A state-based reflex agent maintains internal state using the percept history

needs what is called a model of the environment
knowledge about how it evolves, how actions affect it
also called a model-based reflex agent

A goal-based agent seeks to achieve states which are considered favorable
Unlike reflex agents, makes use of foresight in its decision process

A utility-based agent evaluates future states using a utility function
Seeks to maximize utility of future states

A learning agent can be broken down roughly into four components
A learning element is responsible for improving the agent
A critic evaluates current performance and informs the learning element
(uses a fixed performance standard)
A performance element selects actions
A problem generator suggests exploratory actions

Two of these are reflexive, the next two are planning-based.
Planning agents predict consequences of actions using a transition model.

Agents can vary in the degree to which they deliberate
one extreme: carefully construct a complex plan
another: start with a simple plan and rapidly correct as complications arise

Search Problems

We consider problem-solving agents, which set out to achieve some desirable state
an uninformed search algorithm has no idea of where to look for solutions
relies only upon the problem definition

An agent plan in a search problem will first formulate, then search, then execute

Problem formulation:
simplifies environment, abstracts away unnecessary information
helps organize the behavior of the agent

Components of a well-defined problem:
(1) Initial state
(2) Possible actions: i.e., a successor function $f : \{\text{states}\} \to \{(\text{action, consequent state})\}$
These first two implicitly define a state space, and a state space graph
State space graph has states as nodes and actions as edges
(3) Goal test
(4) Path cost: (e.g. induced by edge costs of a state space graph)
A path is a sequence of actions.

A solution is a path from an initial state to a goal state.
An optimal solution has lowest path cost.

Environments we are considering are static, observable, discrete, and deterministic.

Generalized tree search
envision the search space as a tree (states can be revisited)
nodes have: state, parent-node, action leading to that node, path-cost, and depth
the root of the search tree corresponds to the initial state of the problem

uses a fringe, initialized to contain only the root node
the fringe contains nodes which have been generated but not yet expanded

At each node in the process:
perform goal test on the node in question, if succeeds, return corresponding solution
expand node by i.e. applying successor function to generate new nodes
if no such nodes exists, declare failure
choose which node to analyze next according to some search strategy

Evaluation of a search strategy.
Completeness: whether or not it finds a solution, if a solution exists.
Optimality: if it finds a solution, whether or not such a solution is of lowest cost.
Complexity: space (nodes stored at any given time) or time (total nodes explored)

Factors often used to evaluate the complexity of a given algorithm.
Branching factor b: the maximum number of successors of any node.
Depth d: the minimum depth among goal nodes.
Maximum path length m: can be infinite.

Types of uninformed (aka blind) search strategies
contrast: informed search strategies take advantage of heuristics
cannot solve searches of exponential complexity for all but the smallest problems

Breadth-first search uses a queue (FIFO) as its fringe.
Satifies completeness (if branching factor finite) but not optimality.
Identical space and time complexity (holds all nodes in state until goal is found)
Exponential complexity: $\mathcal{O}(b^{d+1})$
In practice, spacial complexity too hard: very difficult to accrue enough memory

Uniform-cost search orders its fringe by path cost.
As long as each step cost $> \epsilon > 0$, satisfies completeness *and* optimality.
Again, same time and space complexity
Given ϵ minimum action cost, $C*$ optimal solution cost, complexity is $\mathcal{O}(b^{\lceil C*/\epsilon \rceil})$

Depth-first search uses a stack (LIFO) as its fringe
Node storage is not exponential, space complexity is $\mathcal{O}(bm)$.
Backtracking search: nodes store set of successors, search expands only one at a time.
Backtracking search can reduce the space complexity to $\mathcal{O}(m)$.
Worst-case time complexity $\mathcal{O}(b^m)$ where potentially $m \gg d$ and even $m = \infty$.
Complete if $m < \infty$, but not optimal.

Depth-limited search
specifies a depth limit l, and performs a depth-first search up to that limit
if $l < d$, will be incomplete, and if $d < l$, can be non-optimal
has a time complexity of $\mathcal{O}(b^l)$ and a space complexity of $\mathcal{O}(bl)$.
the diameter of the state space: min number of actions between any two states
is a great depth limit, but, we don't necessarily know the diameter, a priori

Depth-first search by iterative deepening
apply depth-limited search with l ranging over $\mathbb{N}$
like DFS, has good memory (spatial complexity) at $\mathcal{O}(bd)$
like BFS, if b is finite will be complete, and if path cost corresponds to depth, optimal

repeated state generation not costly: most nodes in the bottom level
generated nodes: $d(b) + (d-1)b^2 + \cdots + (1)b^d$ (factor of d is node repetition)
hence the time complexity is $\mathcal{O}(b^d)$
better than BFS ($\mathcal{O}(b^{d+1})$): IDS generates no nodes beyond the solution depth
When search space is large and solution depth unknown, depth-first IDS is generally best.
Bidirectional search run simultaneous searches from initial state and goals
Concludes when the searches meet.
Time and complexity $\mathcal{O}(b^{d/2})$, is reduced.
However, requires effective computation of predecessors: often non-trivial/impossible.

Avoiding repetition of states: Graph Search
adds a closed list to tree search: a list of all nodes which have been expanded
the term open list is sometimes used to refer to the fringe
current nodes which match a node on the closed list are discarded
possibly suboptimal, if search methods can reach nodes at a non-optimal cost first
thus uniform cost graph search is optimal but IDS graph search may not be
the closed list increases space requirements, possibly to unfeasibility
behavior of closed list is such that the memory used is proportional to the runtime

Partial Information Search
Sensorless Problems
know consequences of actions and the possible states
can coerce world into a particular state, with some cleverness
uses a belief state: a set of states currently regarded as possible
a solution is a path to a belief state in which all its members satisfy the goal test
this approach can be analogously applied to nondeterministic problems
Contingency Problems
agent can obtain new information from sensors after performing actions
solution: use an unfixed action sequence, with dependencies on percepts received
the action at each node will depend on all percepts received up to that point
agent can act before finding a guaranteed plan
approach referred to as interleaving search and execution
does not need to account for *all* contingencies, simply the ones that occur

Informed Search and Exploration

Informed search uses problem-specific knowledge beyond the definition of the problem
Approach: select nodes for expansion based on an estimate of distance to goal
Use a priority queue ordered by some evaluation function f
Called best-first search: since we order by nodes which seem best
A heuristic $h : \{\text{nodes}\} \to \mathbb{R}^+$ estimates cost of cheapest path to a goal node

Greedy best-first search uses $f = h$
susceptible to false starts, dead ends

same defects as DFS: not optimal, incomplete, worst-case time/space complexity $\mathcal{O}(b^m)$
m the maximum depth of search space

A* search uses $f = g + h$ where g is the path cost to a node
hence f is the estimated cost of cheapest solution through n
for tree search, if h is admissible (never overestimates), A* will be optimal
for graph search, need to ensure the first generated path is optimal
this shall occur if h is additionally consistent/monotone
consistent h satisfies, for nodes n, n' and action $n \xrightarrow{a} n'$, $h(n) \leq c(a) + h(n')$
satisfies a triangle inequality; i.e., h must be a metric on the state space
A* is optimally efficient for a given h: $\nexists$ an optimal, more efficient algorithm
however, nodes in goal contour search space still increase exponentially,
unless $|h - h^*| \leq \mathcal{O}(log(h^*(n)))$ where $h*$ is the true cost
generally $|h - h^* = \mathcal{O}(h^*(n))$ at best
thus, it's often impractical to insist on finding an optimal solution
A* keeps all generated nodes in memory $\rightarrow$ impractical for large-scale problems

Improvements on A* with respect to space complexity

Can try iterative deepening A* (IDA*) using $f = g + h$ rather than d as the cutoff
but this incurs substantial overhead

Recursive best-first search
tracks the f-value of the current best alternative path
winds back to this alternative path once f considered exceeds that stored f
optimal if h is admissible, with space complexity $\mathcal{O}(bd)$
time complexity can vary, depends on h, frequency of path changes
can potentially explore a state multiple times (typical tree-search problem)
stores only the value of f and $\mathcal{O}(bd)$ nodes

MA* (memory-bounded A*) and SMA* (simplified MA*) make use of all memory
description of SMA*: keep expanding until memory is full
once memory is full, replace the worst (by f) node in memory, breaking ties by age
SMA* is complete if there is enough memory to hold the shortest path to a goal
practically, probably best for a graph state space and non-uniform path costs
if too much switching, problems that A* would solve become intractable for SMA*
time $\leftrightarrow$ space tradeoff

Learning to search better
metalevel learning algorithm analyzes current method, seeks improvements

Heuristics
This is where we at.

9/8

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