Intelligent Agents

AIMA, Chapter 2.1-2.2

Outline for today’s lecture

- Intelligent Agents (AIMA 2.1-2)
- Task Environments
- Formulating Search Problems
- Search Fundamentals (if time)

Agents and environments

- An agent is specified by an agent function \( f: P \rightarrow a \) that maps a sequence of percept vectors \( P \) to an action \( a \) from a set \( A \):
  \[
P = [p_0, p_1, \ldots, p_t]
  A = [a_0, a_1, \ldots, a_k]
\]

Agents

- An agent is anything that can be viewed as:
  - perceiving its environment through sensors and
  - acting upon that environment through actuators

  - Human agent:
    - Sensors: eyes, ears, …
    - Actuators: hands, legs, mouth, …

  - Robotic agent:
    - Sensors: cameras and infrared range finders
    - Actuators: various motors

  - Agents include humans, robots, softbots, thermostats, …

Agent function & program

- The agent program runs on the physical architecture to produce \( f \)
  - agent = architecture + program

  - “Easy” solution: table that maps every possible sequence \( P \) to an action \( a \)
    - One small problem: exponential in length of \( P \)

Rational agents II

- Rational Agent: For each possible percept sequence \( P \), a rational agent should select an action \( a \) that is expected to maximize its performance measure.

  - Performance measure: An objective criterion for success of an agent’s behavior, given the evidence provided by the percept sequence.

  - A performance measure for a vacuum-cleaner agent might include e.g. some subset of:
    - +1 point for each clean square in time \( T \)
    - +1 point for clean square, -1 for each move
    - -1000 for more than \( k \) dirty squares
Rationality is not omniscience

- Ideal agent: maximizes actual performance, but needs to be omniscient.
  - Usually impossible.....
  - But consider tic-tac-toe agent...
  - Rationality ≠ Guaranteed Success
- Caveat: computational limitations make perfect rationality unachievable ➔ design best program for given machine resources
- In Economics: “Bounded Rationality” ➔ “Behavioral Economics

Outline for today’s lecture

- Intelligent Agents
  - Task Environments (AIMA 2.3)
- Formulating Search Problems
- (Search Fundamentals)

Task environments

- To design a rational agent we need to specify a task environment
  - a problem specification for which the agent is a solution
- PEAS: to specify a task environment
  - Performance measure
  - Environment
  - Actuators
  - Sensors

PEAS: Specifying an automated taxi driver

Performance measure:
  - ?
Environment:
  - ?
Actuators:
  - ?
Sensors:
  - ?

PEAS: Specifying an automated taxi driver

Performance measure:
  - safe, fast, legal, comfortable, maximize profits
Environment:
  - roads, other traffic, pedestrians, customers
Actuators:
  - steering, accelerator, brake, signal, horn
Sensors:
  - cameras, sonar, speedometer, GPS

PEAS: Medical diagnosis system

- Performance measure: Healthy patient, minimize costs, lawsuits
- Environment: Patient, hospital, staff
- Actuators: Screen display (form including: questions, tests, diagnoses, treatments, referrals)
- Sensors: Keyboard (entry of symptoms, findings, patient’s answers)
The rational agent designer's goal

- Goal of AI practitioner who designs rational agents: given a PEAS task environment,

1. Construct agent function $f$ that maximizes (the expected value of) the performance measure,

2. Design an agent program that implements $f$ on a particular architecture

Environment types: Definitions I

- **Fully observable** (vs. partially observable): An agent’s sensors give it access to the complete state of the environment at each point in time.

- **Deterministic** (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent.
  - If the environment is deterministic except for the actions of other agents, then the environment is **strategic**.

- **Episodic** (vs. sequential): The agent’s experience is divided into atomic “episodes” during which the agent perceives and then performs a single action, and the choice of action in each episode depends only on the episode itself.

Environment types: Definitions II

- **Static** (vs. dynamic): The environment is unchanged while an agent is deliberating.
  - The environment is **semidynamic** if the environment itself does not change with the passage of time but the agent’s performance score does.

- **Discrete** (vs. continuous): A limited number of distinct, clearly defined percepts and actions.

- **Single agent** (vs. multiagent): An agent operating by itself in an environment.

(See examples in AIMA, however I don’t agree with some of the judgments)

Environment Restrictions for Now

- We will assume environment is
  - **Static**
  - **Fully Observable**
  - **Deterministic**
  - **Discrete**

Outline for today’s lecture

- Intelligent Agents
- Task Environments
- **Formulating Search Problems (AIMA, 3.1-3.2)**
- (Search Fundamentals)
Two Approaches to AI

- **Logical representations (Modules 1 & 3)**
  - Dominant BEFORE 1995
  - Relations between entities
    - "Mitch's bicycle is red"
      - (isa B3241 bicycle) (color B3231 red) (owns B3241 P119)
      - (isa P119 person) (name P119 "Mitch")
  - Explicit logical models
    - Search (module 1), Logical inference (module 3)
    - Chess, Sudoku, computer games, …
- **Statistical models (Module 2)**
  - Dominant SINCE 2000
  - Prediction by look-up or by weighted combinations
    - \( P(y=bicycle) = c_0 + c_1 x_1 + c_2 x_2 + c_3 x_3 + \ldots \)
  - Machine Learning, Machine vision, speech recognition, …

Example search problem: 8-puzzle

- **Formulate goal**
  - Pieces to end up in order as shown…
- **Formulate search problem**
  - States: configurations of the puzzle (9! configurations)
  - Actions: Move one of the movable pieces (94 possible)
  - Performance measure: minimize total moves
- **Find solution**
  - Sequence of pieces moved: 3, 1, 6, 3, 1, …

Example search problem: holiday in Romania

- On holiday in Romania; currently in Arad
  - Flight leaves tomorrow from Bucharest
- **Formulate goal**
  - Be in Bucharest
- **Formulate search problem**
  - States: various cities
  - Actions: drive between cities
  - Performance measure: minimize distance
- **Find solution**
  - Sequence of cities; e.g. Arad, Sibiu, Fagaras, Bucharest, …

More formally, a problem is defined by:

**Formulate Search Problem**

1. States: a set \( S \)
2. An initial state \( s_i \in S \)
3. Actions: a set \( A \)
   - \( \forall s \in S \): Activities(s) is the set of actions that can be executed in \( s \), that are applicable in \( s \).
4. Transition Model: \( \forall s \in S \) \( \forall a \in \text{Actions}(s) \): \( \text{Result}(s,a) \rightarrow s' \)
   - \( s' \) is called a successor of \( s \)
   - \( f_s(A, s, a) \) is the step cost, assumed \( \geq 0 \)
   - \( f_s(a) \) is the step cost, assumed \( \geq 0 \)
5. Performance Measure: Path cost
   - Must be additive
   - e.g. sum of distances, number of actions executed, …

**Formulate Goal**

1. Goal test: \( \text{Goal}(s) \)
   - Can be implicit, e.g. \( \text{checkmate}(x) \)
   - \( s \) is a goal state if \( \text{Goal}(s) \) is true

**Find optimal Solution**

- A solution is a sequence of actions from the initial state to a goal state.
- **Optimal Solution**: A solution is optimal if no solution has a lower path cost.
Art: Formulating a Search Problem

Decide:

- Which properties matter & how to represent
  - Initial State, Goal State, Possible Intermediate States
- Which actions are possible & how to represent
  - Operator Set, Actions and Transition Model
- Which action is next
  - Path Cost Function

Formulation greatly affects combinatorics of search space and therefore speed of search

Example: 8-puzzle

- States??
- Initial state??
- Actions??
- Transition Model??
- Goal test??
- Path cost??

Example: 8-puzzle

- States?? List of 9 locations - e.g., [7,2,4,5,-6,8,3,1]
- Initial state?? [7,2,4,5,-6,8,3,1]
- Actions?? (Left, Right, Up, Down)
- Transition Model?? ...
- Goal test?? Check if goal configuration is reached
- Path cost?? Number of actions to reach goal

Hard subtask: Selecting a state space

- Real world is absurdly complex
  - State space must be abstracted for problem solving
- (abstract) State = set (equivalence class) of real world states
- (abstract) Action = equivalence class of combinations of real world actions
  - e.g., Arad → Zerind represents a complex set of possible routes, detours, rest stops, etc
  - The abstraction is valid if the path between two states is reflected in the real world
- Each abstract action should be “easier” than the real problem

Outline for today’s lecture

- Intelligent Agents
- Task Environments
- Formulating Search Problems
- Search Fundamentals (AIMA 3.3)

Useful Concepts

- State space: the set of all states reachable from the initial state by any sequence of actions
  - When several operators can apply to each state, this gets large very quickly
  - Might be a proper subset of the set of configurations
- Path: a sequence of actions leading from one state \( s_0 \) to another state \( s_k \)
- Frontier: those states that are available for expanding (for applying legal actions to)
- Solution: a path from the initial state \( s_i \) to a state \( s_f \) that satisfies the goal test
Basic search algorithms: **Tree Search**

- Generalized algorithm to solve search problems (Review)
  - Enumerate in some order all possible paths from the initial state
  - Here: search through explicit tree generation
    - ROOT= initial state.
    - Nodes in search tree generated through transition model
    - Tree search treats different paths to the same node as distinct

**Review: Generalized tree search**

```plaintext
function TREE-SEARCH(problem, strategy)
    return a solution or failure

Initialize frontier to the initial state of the problem

do
    if the frontier is empty
        then return failure

    choose leaf node for expansion according to strategy & remove from frontier

    if node contains goal state
        then return solution

    else
        expand the node and add resulting nodes to the frontier
```

8-Puzzle: States and Nodes

- A state is a (representation of a) physical configuration
- A node is a data structure constituting part of a search tree
  - Also includes parent, children, depth, path cost g(s)
  - Here node= <state, parent-node, children, action, path-cost, depth>
- States do not have parents, children, depth or path cost!

The EXPAND function
  - uses the Actions and Transition Model to create the corresponding states
    - creates new nodes,
    - fills in the various fields

Problem: Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!

Solution: Graph Search!

- Graph search — Optimal but memory inefficient
  - Simple Mod from tree search: Check to see if a node has been visited before adding to search queue
    - must keep track of all possible states (can use a lot of memory)
    - e.g., 8 puzzle problem, we have 9!/2 = 182K states
Graph Search vs Tree Search

**Function: TREE-SEARCH**(problem) returns a solution, or failure
- Initialize the frontier using the initial state of problem
- loop
  - If the frontier is empty then return failure
  - Choose a leaf node and remove it from the frontier
  - If the node contains a goal state then return the corresponding solution
  - Expand the chosen node, adding the resulting nodes to the frontier

**Function: GRAPH-SEARCH**(problem) returns a solution, or failure
- Initialize the frontier using the initial state of problem
- loop
  - If the frontier is empty then return failure
  - Choose a node and remove it from the frontier
  - If the node contains a goal state then return the corresponding solution
  - Expand the chosen node, adding the resulting nodes to the frontier
  - Add node to the frontier or expanded set

Figure 2.7: An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.