**Lecture 2: Problem Solving with Search**

*9/1/2016*

**Summary**

This lecture is about defining problems and their solutions in a precise way so that we can attempt to solve them computationally (the textbook refers to this type of programs as a “problem-solving agent.”)

**Rational Agent**

* One way to exhibit intelligence is by making good choices while solving problems.
* Agent needs to weigh options and outcomes.
  + Doing so systematically is **search**

Consider a few examples of problems we’d like our agent to solve:

1. Route-finding problems
2. The 8-puzzle or Rubik’s cube
3. N-Queens puzzle (place N queens on an NxN chessboard such that they can’t attack each other)
4. Logic puzzles (e.g., Sudoku, LSAT questions, [The Zebra Puzzle](https://en.wikipedia.org/wiki/Zebra_Puzzle))
5. Some adversarial two-player games (tic-tac-toe, chess, etc.)

How do we set up a common framework so that we can talk about solving all these different kinds of problems in the same way?

* How do we describe “outcomes”?
* How do we describe “options”?

**Hypothesis (Newell, 1980)**

All goal-oriented symbolic activities occur in a **problem** [**state] space**, and intelligence can be characterized as **search in that space**.

**Problem Formulation for Single-State Problems**

* State: a configuration or model of the world.
  + State space: all possible configurations the model can represent
  + Initial state: the configuration at the starting point of the search
  + Goal state: the configuration of a successful outcome
* Actions: operations to transition between states
* Transition Model: applies an action to a state to get a new state
  + The new state is called a successor of the previous state
  + When we transition from state to state to state, we make a path through the state space.
  + A solution path starts at the initial state and ends at a goal state.
* Goal test: has the current state achieved (one of) the goal(s)?
* Path cost: We want to differentiate between different solution paths.
  + Path cost = sum of the costs of all the steps taken in the path
  + To define a step cost, we need to consider a performance metric for our problem (e.g., shortest number of actions? cheapest set of actions? etc.)
    - step cost is represented as a triplet (s1, a, s2), where s1 is some state, a is an action that we can take at s1, and s2 is the successor state after taking action a at state s1.

Question: What abstract data structure would you use to represent the state space?

**Example: route-planning**

* State: current location (Question: What is a “location”?)
  + State space: all possible locations in the area of interest
    - How to represent a location? What are the advantages and disadvantages of your choice?
    - Here, we’ll take a location to specify a point where the agent might have to make another directional choice. For example, our state space could be all street intersections.
  + Init state: start location
  + Goal state: destination
* Actions:  go in one of the possible directions until there is another choice
* Transition Model: Given a location and a direction, return the nearest connected location in the direction.
* Goal test: Current location == destination?
* path cost: depends on what we want to optimize. Examples include:
  + distance traveled
  + “hilly-ness” of the path (e.g., if you don’t want a challenging walk)

**Examples: 8-Puzzle**

* State: current board
  + State space: all possible configurations of the tiles
  + Init state: start board configuration (e.g., random)
  + Goal state: the desired configuration
* Actions: swap the blank space with one of the tiles around it (up, down, left, right)
* Transition model: Given a configuration and a swap direction, swap the blank space with the tile in that direction, and return the resulting configuration.
* Goal test: Current state == desired configuration?
* Path cost: # of steps (each step costs 1)

Notes: it’s easy for the state space to grow exponentially:

* + 8 puzzle: 9!/2 = 181,400 states
  + 15 puzzle: 16!/2 ~ 1013 states
  + 24 puzzle: 1025 states

**Example: N-Queens**

Option #1: incremental formulation

* State: an NxN chessboard with 0-N queens on it
  + State space:
    - Naive (using naive actions): all possible placements of 0-N queens on the board
    - Better (using better actions): 0-N queens on board, one per column in contiguous columns, none attacking each other
  + Init state: an empty board
  + Goal state: all N queens on board, non-attacking
    - Note: this is an implicit goal description. We don’t know what it looks like (it’s what we want to find out!)
* Actions:  add a queen to the board
  + Naive: add the queen to any empty square
  + Better: add the queen to the leftmost empty column such that it’s not attacked by another queen
* Transition model: given a configuration and an action, return the resulting chessboard after performing the action
* Goal test: do we have N queens on board, none attacked?
* path cost: not so interesting for this problem -- we just want to know the goal configuration

Option #2: complete-state formulation

* State:
  + State space:
    - (Naive): any possible configuration of N queens on an NxN chessboard
    - (Better): any possible configuration of N queens on an NxN chessboard, subject to the constraint that there is one queen per column.
  + Init state: a random state in the state space
  + Goal state: none of the N queens attack each other
* Actions: Move an attacked queen somewhere else
  + Naive: move it to anywhere
  + Better: move it within its column to a place of minimum conflict
* Transition model: given a configuration and an action, return the resulting chessboard after performing the action
* Goal test: do we have N queens on board, none attacked?
* path cost: not so interesting for this problem -- we just want to know the goal configuration

Note: whether we go with Option #1 or Option #2, the important thing to keep in mind is to avoid useless extra states. (Naive state space for option 1 is on the order of 1014 while the better representation requires just thousands of states.)

**In class exercise (Water Jugs)**

scenario

“You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a tap that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?”. (cf. [Rich & Knight, 1991](http://www.amazon.com/Artificial-Intelligence-Elaine-Rich/dp/0070522634))

* State: amount of water in (3-gallon jug, 4-gallon jug)
  + State space: all possible contents of the jugs
  + Init state: (0, 0)
  + Goal state: (0, 2)
* Actions:
  + A1: Fill 4-gallon jug from tap
  + A2: Fill 3-gallon jug from tap
  + A3: Dump 4-gallon jug on the ground
  + A4: Dump 3-gallon jug on the ground
  + A5: Fill 4-gallon jug from 3-gallon jug
  + A6: Fill 3-gallon jug from 4-gallon jug.
* Transition Model:
  + If current state is (x, y):
    - If can perform A1, the new state is (x, 4)
    - If can perform A2, the new state is (3, y)
    - If can perform A3, the new state is (x, 0)
    - If can perform A4, the new state is (0, y)
    - If can perform A5, the new state is (x-d, y+d), where d = min(x, 4-y)
    - If can perform A6, the new state is (x+d, y-d), where d = min(3-x, y)
* Goal test: Current state == goal state?
* path cost: # of actions

Take-home exercise: Draw the state space. How many possible states does it have? (At a first glance, there seems to be 4x5=20 states; however, if you enumerate all possible states reachable from the init state, you’ll see that some configurations are not possible. (I counted 6 of those. Do you agree?)

**Corresponding Readings**

* Chapter 3 of Russell, S. and Norvig, P. (2010). Artificial Intelligence: A Modern Approach. 3rd ed. Prentice Hall.