Deliberative Agents

1 Thinking Ahead

We’ve assumed that our agents implement, in some form, a sense/plan/act cycle. But so far, the plan phase has been quite simple. In particular, our agents have not done any thinking ahead. In general, to build a more intelligent agent, we require that the plan phase be more deliberative. To choose between actions, agents must think through the consequences of actions ‘in their heads’ prior to execution. It will often be better still to consider whole sequences of actions. (For example, consider the way players think ahead in chess.)

Thinking ahead is a form of simulation: trying out an action or a sequence of actions on a mental representation prior to executing the action(s) in the actual world. Let me stress: during deliberation (simulation), the effects of chosen actions are used to update only the model of the world. There is no execution of actions on the environment during this phase. Only once the plan phase has been completed and an action that we hope will be effective has been chosen, do we move on to the act phase of the cycle and actually execute the action.

Class exercise. Can you give more precise reasons why this kind of thinking ahead is advantageous: what can go wrong if you don’t think ahead?

Class exercise. Are there times when thinking ahead is disadvantageous: what can go wrong if you do think ahead?

As mentioned above, it is often going to be the case that, during the plan phase, the agent investigates whole sequences of actions. But this gives us at least two ways of constructing our agent. On the one hand, the agent could sense the current state of the world, run its simulation to decide on a sequence of actions that would transform the world from its current state to a goal state, hand this sequence over to the execution phase, and then execute the whole sequence. Such an agent would need to work through the sense/plan/act cycle only once:
On the other hand, the agent could sense the current state of the world, run its simulation to decide on a sequence of actions that would transform the world from its current state to a goal state, hand only the first of the actions in the sequence over to the execution phase, and then execute this one action. It would then repeat this sense/plan/act cycle. So, in this scenario, although whole sequences of actions are being investigated, only the first action in the sequence is being executed for real:

Class exercise. *The second approach appears to be wasteful. But the first approach is suitable only for certain environments. What kinds of environments?*

Of course, these two approaches are extremes. One can imagine intermediate approaches where a handful of actions from the action sequence are executed before the next sense/plan/act cycle.

In what follows, I shall write these notes as if we were using the first approach. This is done for no reason other than to simplify the explanation.
2 State Space Search

We’ve been using phrases such as thinking ahead/ planning ahead/ deliberation/ simulation. But the way we implement this process is using search.

Unfortunately, the word ‘search’ has two meanings in AI (and, to a lesser extent, in Computer Science). (Actually, they’re not really different, but your understanding might be improved if we keep them separate.)

There is its ‘traditional’ meaning: where we are seeking some item in a data structure (e.g. linear search or binary search of an array, linked list, file or database table). Obviously, in AI, we sometimes need to carry out such operations, and so we sometimes use the word ‘search’ with this ‘traditional’ meaning.

The other meaning is the more common one in AI, where we have a graph (nodes and edges) and we are searching for (trying to find) a path (sequences of connected edges) in the graph from one node to another.

Deliberation, then, is about searching for paths in graphs. In this case, we’re using directed graphs (where each edge has a direction, indicated by an arrowhead). The nodes of the graph represent possible states the world can be in. Each edge between two nodes represents execution of an action, transforming one state to another. One of the nodes represents the initial state of the world, the start state. One or more nodes represent goal states. The task is to find a path from the node labelled by the start state to one of the nodes labelled by goal states.

Such a graph, of states reachable by sequences of actions from some start state, is called a state space.

But . . .

In Computer Science and Mathematics, such graphs are specified quite explicitly. Mathematically, you are given two finite sets: the set of nodes and the set of edges. Implementationally, the graph will typically be stored in memory as a node-and-pointer data structure.

In AI, the graphs (state spaces) can be very large. (Very very large.) (Some people even consider the possibility of graphs with an infinite number of states. But we won’t.) Therefore, we do not (and maybe cannot) specify them explicitly. We use an implicit specification. State spaces are specified by giving:

- the start state;
- the set of operators for transforming states to other states; and
- the goal condition that can detect whether a state is a goal state or not.

While the graph itself is not usually explicitly given (because it’s generally too large), it is, in principle, possible to make it explicit from this implicit
specification. Indeed, as search proceeds, the search algorithm will explicitly construct parts of the graph in memory.

Note that it is also common to associate numbers with each operator, representing the cost or benefit (in terms of time, money, energy, or whatever) of using the corresponding action. We can then compute the total cost of a path in the graph by summing the costs of the edges along that path. This path cost function is typically called $g$. (Unfortunately, the activation function in a neural network was also called $g$. These are two completely distinct uses of the letter $g$. Don’t muddle them up.)

3 The 8-Puzzle

Here’s an example of a state space. It’s a toy example, but it’s a huge space. If toy examples give such huge spaces, imagine what real world problems might be like.

In the 8-puzzle, 8 uniquely-numbered tiles sit in a $3 \times 3$ grid. The task for an agent is to find a sequence of tile-sliding actions that transforms some initial configuration to some goal configuration. An obvious representation for the states is a $3 \times 3$ array of integers.

Let’s specify a state space for this puzzle.

- The start state might be:
  
  \[
  \begin{array}{ccc}
  2 & 8 & 3 \\
  1 & 6 & 4 \\
  7 & 5 \\
  \end{array}
  \]

- It’s tempting to think that there are 32 operators: move 1 up, move 1 left, move 1 right, move 1 down, move 2 up, . . . . But, more compactly, if we can swallow the seeming absurdity of it, we need only 4 operators: for moving the blank:

  * if blank is not at top edge then move it up
  * if blank is not at left edge then move it left
  * if blank is not at right edge then move it right
  * if blank is not at bottom edge then move it down

- The goal state might be:
  
  \[
  \begin{array}{ccc}
  1 & 2 & 3 \\
  8 & 4 \\
  7 & 6 & 5 \\
  \end{array}
  \]

In the lecture, we’ll make explicit a small part of this implicitly-specified graph.

This state space has $9! = 362,880$ states.
4 The Water Jugs Problem

Here’s another state space example.

An agent has a 4-gallon and a 3-gallon jug. Neither jug has any measuring markers on it. Also available are a tap and a drain, and nothing else. The jugs are initially empty. The goal is to get exactly 2 gallons of water into the 4-gallon jug.

An obvious representation for the states is a pair of integers $\langle x, y \rangle$, where $x$ is the amount of water in the 4-gallon jug ($x \in \{0, 1, 2, 3, 4\}$) and $y$ is the amount of water in the 3-gallon jug ($y \in \{0, 1, 2\}$).

- The start state is represented by $\langle 0, 0 \rangle$.
- The operators are:
  1. if $x < 4$ then $\langle 4, y \rangle$
  2. if $y < 3$ then $\langle x, 3 \rangle$
  3. if $x > 0$ then $\langle 0, y \rangle$
  4. if $y > 0$ then $\langle x, 0 \rangle$
  5. if $x + y \geq 4 \land y > 0$ then $\langle 4, y - (4 - x) \rangle$
  6. if $x + y \geq 3 \land x > 0$ then $\langle x - (3 - y), 3 \rangle$
  7. if $x + y \leq 4 \land y > 0$ then $\langle x + y, 0 \rangle$
  8. if $x + y \leq 3 \land x > 0$ then $\langle 0, x + y \rangle$
- The goal states are any that match the pattern $\langle 2, n \rangle$.

In the lecture, we’ll make explicit a small part of this implicitly-specified graph. This state space has only 20 states, but there are numerous cyclic paths through the graph.