Deliberation

1 Thinking Ahead

We've assumed that our agents implement, in some form, a sense/plan/act cycle. But so far, because we've been looking mostly at reactive agents, the plan phase has been quite simple. 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.)

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

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

Here's another interesting (if somewhat subtle) way of thinking about this. If a reactive agent (or some other agent that doesn't think ahead) is to be effective (e.g. achieve some goal), then it will take a lot of effort to come up with its action function (whether we come up with it through human design, through evolution or through learning). Its action function must anticipate all the possible situations the agent might find itself in and ensure that, in all those possible situations, the right action (for achieving the goal) is chosen. But, agents that can plan ahead can reduce design effort (or evolution or learning). Making sure that the right action is chosen in a myriad of different circumstances now becomes the responsibility of the agent and doesn't require superhuman foresight on the part of the designer (or evolution or learning). Effort is moved from design-time to run-time.

2 Agents that Think Ahead

Thinking ahead is a form of simulation, and it requires the following:

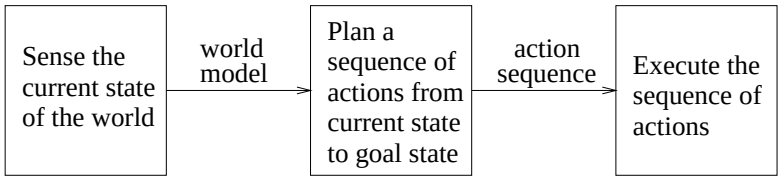
- A mental *model* of the initial state of the world, on which the effects of the actions can be simulated.
- The action repertoire of the agent. This will typically take the form of a *set of operators* (e.g. production rules) that specify
  - *preconditions* that need to be true to enable an action to be executed; and
  - *effects* of the action on the world.
- A *goal condition*, which will specify what the world needs to be like for the agent to have achieved its goal.

You have to now clearly draw distinctions in your mind between what we were looking at before (with reactive agents) and what we're looking at now.

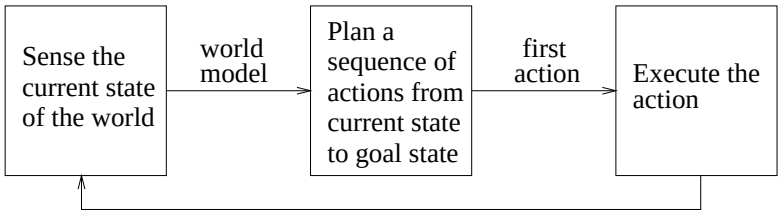
First, note the difference between the rules we use for reactive agents and the operators we use for deliberative agents. In a reactive agent, the conditions in the rules have to be carefully crafted to state the exact circumstances in which we would want the action to be executed. It's not just a question of saying when the action *could* be executed, but when it *should* be executed. In deliberative agents, the preconditions of the operators state only the circumstances in which the action could be executed. For example, in my wall-following agent, I wrote a rule along the lines of 'if there's nothing in front of the agent but there is a wall to the right of the agent, then move forward'. This rule tries to ensure that the move action is only executed when it will help the agent to achieve its wall-following goal. But, in a deliberative agent, the move operator might simply state 'if there's nothing in front of the agent, then you can move forward'. This operator simply gives the necessary conditions for a move action to be possible. It is up to the agent to decide when it would be best to use this action in pursuit of some goal.

Second, note that in the plan phase of the sense/plan/act cycle, where the agent is doing its 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.

Third, it is often going to be the case that, during the plan phase, the agent investigates whole sequences of actions. (As we've discussed, agents generally can't choose their next action without thinking through subsequent 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:



Question. *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.

We are also, for the moment, going to assume that our models of the world are iconic. More complicated algorithms are needed for logical representations, so we postpone looking at them for a while.

Finally, we're going to assume (for no reasons again other than simplicity and concreteness) that our models of the world are always correct and fully-specified (e.g. there will be no 'unknowns' as there were in the lawn mower example).

3 Search and AI

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 (e.g. searching for the first rule in a production system whose condition is true), 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.

This second kind of search pervades AI. We're about to use it for deliberation in an agent's plan phase. We have already been using it (implicitly) in earlier lectures: GAs are searching for fit agents; and neural net learning algorithms are searching for sets of weights that minimise error. (Re-read the back-prop lecture and note the phrase *gradient descent search*). We'll be using it elsewhere too: other forms of learning that we'll look at involve searching for good descriptions of their input data; and many forms of reasoning involve searching for arguments that chain together premises and conclusions.

Reflecting the importance of search, one of my own definitions of AI is:

> *AI studies:*
>
> - *the application of search to knowledge; and*
> - *the application of knowledge to search.*

So, as we proceed, bear in mind that the search techniques we describe will crop up again in other contexts in the course.

4 State Space Search

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 an operator transforming one state to another. One of the nodes, the *start node*, represents the initial state of the world. One or more *goal nodes* represent goal states. The task is to find a path from the start node to one of the goal nodes.

Such a graph, of states reachable by sequences of actions from some initial 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 node*, which is labelled by a representation (in our case, for the moment, an iconic representation) of the initial state of the world;
- the *set of operators* for transforming states to other states; and
- the *goal condition* that can detect whether a node in a graph represents a goal state.

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.

5 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 3×3 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 (iconic) representation for the states is a 3×3 array of integers.

Let's specify a state space for this puzzle.

- The initial state might be:

2	8	3
1	6	4
7		5

- 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:

1	2	3
8		4
7	6	5

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

This state space has $9! = 362,880$ states.

6 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 (iconic) 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 ($y \in \{0, 1, 2, 3\}$).

- The initial 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** $x + y \geq 4 \wedge y > 0$ **then** $\langle 4, y - (4 - x) \rangle$
  6. **if** $x + y \geq 3 \wedge x > 0$ **then** $\langle x - (3 - y), 3 \rangle$
  7. **if** $x + y \leq 4 \wedge y > 0$ **then** $\langle x + y, 0 \rangle$
  8. **if** $x + y \leq 3 \wedge x > 0$ **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.

7 Real State Spaces

In the lecture, we'll briefly discuss some of the many real-world tasks that can be construed as state-space search. They include, e.g., route finding, cargo loading, VLSI layout and automatic assembly.

Exercises

Hint (and rant). In these exercises you are asked for iconic representations for states. I am sick (!) of student answers to these questions involving arrays. Just because the 8-tiles puzzle uses arrays, doesn't mean that all answers use arrays. Re-read the notes from the previous lecture to remind yourself what an iconic representation is. Use arrays only if they seem like a suitable iconic representation. Otherwise consider the numerous other possiblilities: pairs, triples, and more generally, n-tuples; lists; sets; bags; stacks; queues; trees, ...

1. Look at the operators in the Water Jugs Problem. The first operator can be paraphrased as: *if the 4-gallon jug is not full, then fill it from the tap*. Paraphrase the other 7 operators. (Very literal paraphrases such as 'if the amount in the 4-gallon jug plus the amount in the 3-gallon jug is less than or equal to 4,...' are not acceptable!)

2. (*Past exam question*) **The Towers of Hanoi Problem**

   There are 3 poles, and there are five disks of different diameters each having a hole in its centre so that it can be slotted onto the poles. The disks are all on the first of the poles and must be moved to the third of the poles. Only one exposed disk may be moved at a time. At all times, the disks on a pole will be in decreasing order of size (smallest on the top).

   While there are straightforward (recursive and non-recursive) algorithms to solve this problem, in this question you formulate it as an AI search problem. You must develop the problem representation on which search algorithms could be used. (Extra marks will be gained for precision.)

   (a) Give an iconic representation for the *states*

   (b) Give the *initial state*.

   (c) Give the *goal state*.

   (d) Give the *operators*. (The easiest formulation uses six operators. Showing one of the six and giving a sentence that explains briefly how the rest differ will suffice.)

3. **The Towers of Brahma Problem**

   There are 3 poles, arranged in a row; there are five disks of different diameters, each having a hole in its centre so that it can be slotted onto the poles. The disks are all on the first of the poles and must be moved to the third of the poles. Only one exposed disk may be moved at a time. At all times, the disks on a pole will be in decreasing order of size (smallest on the top). When moving a disk, it can only be moved to an adjacent tower, i.e. a disk cannot be moved from the leftmost of the 3 poles to the rightmost (or vice versa) in a single move.

   If you were formulating this problem as an AI search problem, how would your formulation differ from the one you gave for the Towers of Hanoi, above?

4. (*Very, very hard!*) Consider the following scenario.

   Five men and five women have just been married on a tropical island. All now wish to return to the mainland. A boat, which can carry at most three people, is available to ferry the newly-weds back to the mainland. The boat must be crewed by at least one person, male or female, on every trip it makes.

   Libidos are at full stretch and the couples have agreed that no woman should at any point find herself in the company of the other men (even if those men are with their wives!) unless her husband is with her.

   You must develop the problem representation on which search algorithms could be used. (Extra marks will be gained for precision.)

   (a) Give an iconic representation for the states.
   Give the initial state.
   Give the goal state.

   (b) Give the operators.