**Lecture 2 (Intelligent agents)**

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**. A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators. A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

We use the term **percept** to refer to the agent’s perceptual inputs at any given instant. An agent’s **percept sequence** is the complete history of everything the agent has ever perceived. In general, *an agent’s choice of action at any given instant can depend on the entire percept* *sequence observed to date, but not on anything it hasn’t perceived.* By specifying the agent’s choice of action for every possible percept sequence, we have said more or less everything there is to say about the agent. Mathematically speaking, we say that an agent’s behavior is described by the **agent function** that maps any given percept sequence to an action.

We can imagine *tabulating* the agent function that describes any given agent; for most agents, this would be a very large table—infinite, in fact, unless we place a bound on the

length of percept sequences we want to consider. Given an agent to experiment with, we can,

in principle, construct this table by trying out all possible percept sequences and recording

which actions the agent does in response. The table is, of course, an *external* characterization

of the agent. *Internally*, the agent function for an artificial agent will be implemented by an

**agent program**. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.



To illustrate these ideas, we use a very simple example—the vacuum-cleaner world shown in Figure 2.2. This world is so simple that we can describe everything that happens; it’s also a made-up world, so we can invent many variations. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square.



Looking at Figure 2.3, we see that various vacuum-world agents can be defined simply by filling in the right-hand column in various ways. The obvious question, then, is this: *What* *is the right way to fill out the table?* In other words, what makes an agent good or bad, intelligent or stupid? We answer these questions in the next section.

**The concept of rationality**

A **rational agent** is one that does the right thing—conceptually speaking, every entry in the table for the agent function is filled out correctly. Obviously, doing the right thing is better than doing the wrong thing, but what does it mean to do the right thing?

We answer this age-old question in an age-old way: by considering the *consequences* of the agent’s behavior. When an agent is plunked down in an environment, it generates asequence of actions according to the percepts it receives. This sequence of actions causes theenvironment to go through a sequence of states. If the sequence is desirable, then the agenthas performed well. This notion of desirability is captured by a **performance measure** thatevaluates any given sequence of environment states.

Obviously, there is not one fixed performance measure for all tasks and agents; typically,

a designer will devise one appropriate to the circumstances. This is not as easy as it sounds.

Consider, for example, the vacuum-cleaner agent from the preceding section. We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift.

With a rational agent, of course, what you ask for is what you get. A rational agent can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor,

then cleaning it up again, and so on. A more suitable performance measure would reward the

agent for having a clean floor. For example, one point could be awarded for each clean square at each time step (perhaps with a penalty for electricity consumed and noise generated). *As* *a rule, it is better to design performance measures according to what one actually* *wants in the environment, rather than according to how one thinks the agent should behave.* Even when the obvious pitfalls are avoided, there remain some knotty issues to untangle.

For example, the notion of “clean floor” in the preceding paragraph is based on average cleanliness over time. Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time while the other cleans energetically but

takes long breaks. Which is preferable might seem to be a fine point of janitorial science, but

in fact it is a deep philosophical question with far-reaching implications. Which is better—

a reckless life of highs and lows, or a safe but humdrum existence? Which is better—an

economy where everyone lives in moderate poverty, or one in which some live in plenty

while others are very poor? We leave these questions as an exercise for the diligent reader.

**Rationality**

What is rational at any given time depends on four things:

• The performance measure that defines the criterion of success.

• The agent’s prior knowledge of the environment.

• The actions that the agent can perform.

• The agent’s percept sequence to date.

This leads to a **definition of a rational agent**:

*For each possible percept sequence, a rational agent should select an action that is expected*

*to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

Consider the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the

other square if not. Is this a rational agent? That depends! First, we need to say what the performance measure is, what is known about the environment, and what sensors and actuators the agent has. Let us assume the following:

• The performance measure awards one point for each clean square at each time step, over a “lifetime” of 1000 time steps.

• The “geography” of the environment is known *a priori* but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The Left and Right actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is.

• The only available actions are Left , Right, and Suck.

• The agent correctly perceives its location and whether that location contains dirt.

We claim that *under these circumstances* the agent is indeed rational; its expected performance is at least as high as any other agent’s.

One can see easily that the same agent would be irrational under different circumstances.

For example, once all the dirt is cleaned up, the agent will oscillate needlessly back and forth; if the performance measure includes a penalty of one point for each movement left

or right, the agent will fare poorly. A better agent for this case would do nothing once it is

sure that all the squares are clean. If clean squares can become dirty again, the agent should

occasionally check and re-clean them if needed. If the geography of the environment is unknown, the agent will need to explore it rather than stick to squares A and B.

**Building a rational agent**

**Specifying the task environment**

In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent’s actuators and sensors. We group all these under the heading of the **task environment**. For the acronymically minded, we call this the **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors) description. In designing an

agent, the first step must always be to specify the task environment as fully as possible. The vacuum world was a simple example; let us consider a more complex problem: an automated taxi driver. We should point out, before the reader becomes alarmed, that a fully automated taxi is currently somewhat beyond the capabilities of existing technology. The full driving task is extremely *open-ended*. There is no limit to the novel combinations of circumstances that can arise—another reason we chose it as a focus for discussion. Figure 2.4 summarizes the PEAS description for the taxi’s task environment. We discuss each element in more detail in the following paragraphs.



First, what is the **performance measure** to which we would like our automated driver to aspire? Desirable qualities include getting to the correct destination; minimizing fuel consumption and wear and tear; minimizing the trip time or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.

Next, what is the driving **environment** that the taxi will face? Any taxi driver must deal with a variety of roads, ranging from rural lanes and urban alleys to 12-lane freeways. The roads contain other traffic, pedestrians, stray animals, road works, police cars, puddles, and potholes. The taxi must also interact with potential and actual passengers. There are also some optional choices. The taxi might need to operate in Southern California, where snow is seldom a problem, or in Alaska, where it seldom is not. It could always be driving on the right, or we might want it to be flexible enough to drive on the left when in Britain or Japan. Obviously, the more restricted the environment, the easier the design problem.

**The structure of agents**

So far we have talked about agents by describing *behavior*—the action that is performed after any given sequence of percepts. Now we must bite the bullet and talk about how the insides work. The job of AI is to design an **agent program** that implements the agent function— the mapping from percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators—we call this the **architecture**:

*agent* = *architecture* + *program*.

Obviously, the program we choose has to be one that is appropriate for the architecture. If the program is going to recommend actions like *Walk*, the architecture had better have legs. The architecture might be just an ordinary PC, or it might be a robotic car with several onboard computers, cameras, and other sensors. In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program’s action choices to the actuators as they are generated.

**Agent programs**

The agent programs all have the same skeleton: they take the current percept as input from the sensors and return an action to the actuators. Notice the difference between the agent program, which takes the current percept as input, and the agent function, which takes the entire percept history. The agent program takes just the current percept as input because nothing more is available from the environment; if the agent’s actions need to depend on the entire percept sequence, the agent will have to remember the percepts. For example, Figure 2.7 shows a rather trivial agent program that keeps track of the percept sequence and then uses it to index into a table of actions to decide what to do. The table—an example of which is given for the vacuum world in Figure 2.3—represents explicitly the agent function that the agent program embodies. To build a rational agent in this way, we as designers must construct a table that contains the appropriate action for every possible percept sequence.



It is instructive to consider why the table-driven approach to agent construction is doomed to failure. Let P be the set of possible percepts and let T be the lifetime of the agent (the total number of percepts it will receive). The lookup table will contain $\sum\_{t=1}^{T}|P|^{t}$ entries. Consider the automated taxi: the visual input from a single camera comes in at the rate of roughly 27 megabytes per second (30 frames per second, 640×480 pixels with 24

bits of color information). This gives a lookup table with over 10250,000,000,000 entries for an

hour’s driving. Even the lookup table for chess—a tiny, well-behaved fragment of the real

world—would have at least 10150 entries. The daunting size of these tables (the number of

atoms in the observable universe is less than 1080) means that (a) no physical agent in this

universe will have the space to store the table, (b) the designer would not have time to create

the table, (c) no agent could ever learn all the right table entries from its experience, and (d)

even if the environment is simple enough to yield a feasible table size, the designer still has

no guidance about how to fill in the table entries.

Despite all this, *does* do what we want: it implements the desired agent function. The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table. We have many examples showing that this can be done successfully in other areas: for example, the huge tables of square roots used by engineers and schoolchildren prior to the 1970s have now been replaced by a five-line program for Newton’s method running on electronic calculators. The question is, can AI do for general intelligent behavior what Newton did for square roots? We believe the answer is yes.

We outline four basic kinds of agent programs that embody the principles underlying almost all intelligent systems:

• Simple reflex agents;

• Model-based reflex agents;

• Goal-based agents; and

• Utility-based agents.

Each kind of agent program combines components in particular ways to generate actions.

