Agents in the Real World

Computational Models in Artificial Intelligence and Cognitive Science

Matthew Stone
mdstone@cs.rutgers.edu

September 4, 2003

1 Motivation and Overview

The mid-twentieth century saw the introduction of a new general model of processes, COMPUTATION, with the work of scientists such as Turing, Chomsky, Newell and Simon.[1] This model so revolutionized the intellectual world that the dominant scientific programs of the day—spearheaded by such eminent scientists as Hilbert, Bloomfield and Skinner—are today remembered as much for the way computation exposed their stark limitations as for their positive contributions.[2] Ever since, the field of Artificial Intelligence (AI) has defined itself as the subfield of computer science dedicated to the understanding of intelligent entities as computational processes.

Now, drawing on fifty years of results of increasing breadth and applicability, we can also characterize AI research as a concrete practice: an ENGINEERING APPROACH to constructing COMPUTATIONAL ARTIFACTS that act in the REAL WORLD. Such artifacts are known as AGENTS, so we can also describe AI as the practice of agent design. This practice boasts a wide range of successes, in domains from human-computer interaction to robotics to electronic commerce and services, and AI research increasingly emphasizes the techniques and insights required to capture sophisticated behavior in effective applications. With these practical successes, AI may seem to have retreated from the analysis of human intelligence. In fact, however, the view of human intelligence as a computational process now guides empirical and theoretical research throughout academia, in philosophy, linguistics, psychology, and even neuroscience. These diverse scientists continue to frame their explanations of human intelligence in computational terms, and they continue to build on AI research.

However, research in AI and elsewhere has made it clear that we cannot explain how people can act so successfully in such a complex world MERELY by appeal to the generality and power of computation. Rather, as this volume[3] attests, a complete account of human intelligence must encompass further principles that characterize human biology, human environments, and human abilities. For, as the cognitive sciences document in stunning complexity, our natural computation doesn't just endow us with general reasoning: it attunes us to a specific range of complex but significant relationships in the world, and allows us to exploit these relationships to act effectively. Nevertheless, AI can still offer unique insights into human intelligence, by helping to characterize the computational mechanisms and the real-world knowledge that must underlie our own complex, adaptive behavior; and many AI researchers are proud to offer what contributions they can.

In this chapter, I will first set out, in broad strokes, the activities of modeling, implementation and evaluation that distinguish the practice of agent design. I continue by illustrating these activities in more depth as they play out in the most straightforward case of decision analysis for agent design. The illustration not only showcases the central elements of AI practice, but also highlights the insight that AI practice can give into the problems inherent in computational systems that act effectively in the world. AI research has discovered important tradeoffs in agent design and developed a range of effective practical techniques in response. Natural computation must respond to these tradeoffs too, so I encourage you to view AI not only as a toolkit for innovative practical system-building but also as a inspiration to new research into the computational problems of acting in our world.

2 A General Description of Agents

2.1 Agents Take Real-world Actions Based on Real-world Data

In thinking of AI as an engineering approach to building computational artifacts that act in the real world, it is easy at first to overlook the dramatic constraints that the real world imposes on agents, and indirectly on engineers who build agents. But consider how much freedom in design we find in more typical computer programming—in structuring classes and methods in object-oriented programming, for instance.

On the one hand, we can often expect to stipulate what inputs the program will accept, in an unambiguous and restrictive specification. Thus, a compiler or interpreter works with texts described by a formal grammar; operating-systems modules work with the elements of a particular protocol or programming interface; and even a user interface channels its communication with its human user through a small and predefined set of menus, boxes and other widgets. In circumscribing input this way, we presuppose that other people will respect these artificial limits: they must take responsibility to write syntactic code, to implement applications that conform to a systems protocol, to learn to use a graphical interface.

On the other hand, we can typically expect to frame the output of the program in terms of extremely flexible suites of low-level actions. Look at the sequences of microprocessor instructions delivered by a compiler, the storage and communications primitives beneath an operating system, or the graphics commands of a windowing system for a user interface. These actions are carefully designed and robustly implemented to achieve useful effects reliably and easily. Such resources are never perfect, but in practice, programmers can expect to overcome any unexpected results or missing functionality with simple workarounds, not global analysis.

These expectations make for an attractive and useful division of labor. The design of a computer program becomes an exercise in internal organization—a matter of carving up an overall task into subproblems in an elegant, comprehensible, extensible way and of realizing lean, encapsulated modules to solve these subproblems. Such tasks are challenging but sharply delineated.

These idealizations collapse with a commitment to the real world, to an input environment and output effects that precede our implementation and exhibit no overarching design. In an agent, a designer can no longer stipulate any artificial invariants that inputs must respect; the inputs must be those that nature provides. In an agent, a designer can no longer presume a general repertoire of actions with robustly-designed links to desired effects; the consequences of actions unfold naturally, through the inherent dynamics of a broader world. As if this weren't enough, the complexity and unpredictability of the real world seems to complicate the internal organization of an agent, too. It is fantastically difficult to carve small and interesting tasks out of the world and solve them in a constrained way, the way you would carve up the problems induced in typical computer programs and write separate modules to handle them. Indeed, some AI researchers have suggested that real-world problems are so complex and interdependent that any sufficiently interesting problem will effectively involve a solution to all of them. (This leads to the gag concept of an AI-complete problem.)

The difficulties of agent design come as no surprise with the picturesque tasks that sometimes motivate AI research; after all these are tasks that are only now starting to afford suggestive implementations. One such benchmark is an agent design to control a physical robot running tours or deliveries in a populated space.[4] Here, inputs might include the readings of real sensors from the environment: optical cameras, laser and sonar rangefinders, perhaps a global-positioning receiver. These readings are always more or less noisy, and sometimes fail altogether. Outputs, meanwhile, might take the form of voltages to motors (on wheels, for example); the success or failure of such an action will depend on such ephemera as the location of the robot itself, the location of people and other obstacles in the robot's surroundings, and even the condition of the floor. The gap between inputs and outputs calls for an implementation whose every action responds to an imposing range of considerations. The agent must draw on background information, such as maps of the environment; it must interpret its current sensed data as mirroring or diverging from the real state of the world; it must predict and weigh the range of effects a choice might have in its specific situation.

Despite their almost science-fiction flavor, these ambitious benchmarks do call attention to practical problems that arise much more generally in cutting-edge computer systems. The use of real-world actions and real-world data, for example, is now an increasingly common requirement. When internet software assesses the content of web pages written by people for people, it must deal with the human idiosyncracies of style, sloppiness and error. When ubiquitous computation is embedded not in a predefined computational infrastructure but an open-ended physical environment, inputs and outputs involve inevitable uncertainties and indirections. Any commitment to the real world, no matter how constrained, brings with it the ensemble of perspectives and challenges to which AI research responds.

Real-world tasks give AI a focus on MODELING, describing the real world by a mathematical approximation. A model, as an approximation of its object, acknowledges the open-ended texture of reality; there are always finer distinctions to be drawn, wider-ranging interactions to capture. An effective design allows an agent to respond robustly across a range of possible environments despite these approximations—a good model can capture just those features and regularities that an agent depends on in the environment for its success. But even a good model suggests possible refinments. Pursuing them not only guides the development of more successful and flexible agents, but also, in the best case, can challenge our understanding of the rich world in which we too must act.

Mathematics imbues a model with precision. With the methods of logic, probability and statistics, we can definitively articulate the information about the world our model can offer, and the assumptions about the world our model embodies. In so doing, we lay essential groundwork for the implementation and evaluation of an agent that exploits just this information, but depends on just these assumptions, to solve its real-world tasks.

2.2 Agents Are Computational Processes

We live in a world so pervaded by silicon, in all its ramifications, that it is easy to forget that there are alternatives to computational processes. But of course many artifacts—from clocks to calculators and beyond—could readily have assemblies of gears and escapements, switches and levers, in place of the chips we now typically find; not so long ago, in fact, the mechanical design would have been inevitable. Over time, many fields have taken up the description of physical systems that generate sophisticated real-world behavior in response to messy real-world data. These fields have developed fundamental mathematical tools which Moore's law increasingly brings within the sphere of a computational approach: a case in point is the discrete-time Kalman filter (from control theory in electrical engineering), a probabilistic framework for optimally reconciling a noisy prediction from the past and a noisy datum from the present.[5] Such contrasts impel us to be precise about computation; it is, after all, perhaps the oldest and strongest constant of AI practice. I follow Allen Newell's answer.[6]

To say that an agent is a computational artifact means that it is a PHYSICAL SYMBOL SYSTEM. In this context, a SYMBOL is some discrete, atomic element which STANDS FOR something in the real world. The term originates with the nineteenth-century American philosopher Peirce, who defined a symbol as any object that has an arbitrary conventional meaning.[7] In a concrete implementation in AI, the symbols happen to be sequences of bits in an agent's digital memory. The designer of an agent typically determines what each symbol stands for, by describing an intended correspondence between symbols and the elements of the agent's environment. For example, a designer may invent a symbol to name an individual object in the environment, or may use symbols in a more general and abstract way, to name some property that objects may or may not enjoy or some possible state of affairs that may or may not really be the case. Symbolic programming languages are available to help mediate this correspondence, and designers can also formalize the correspondence in their mathematical model of an agent's environment.

In a physical symbol system, occurrences of these symbols are assembled together into REPRESENTATIONS: complexes of symbols that correspond to statements about the real world. The correspondence endows the state of the agent—otherwise just an array of ones and zeros—with intelligible meaning. The agent's representations may spell out facts that the agent knows, goals that the agent has for its environment, or intentions to which the agent is committed. These representations give the agent meaningful reasons to act in its environment in a specified way—reasons to act that can themselves take the form of representations. Meanwhile, when we link an agent's representations further to a formal model of its environment, we can use our understanding of the model to inform our understanding of the agent's behavior. An agent's representations, and its behavior, can fit the world only as much as the model that guides its design.

Now, this representational understanding of computation is in a sense superfluous: meaning is actually irrelevant to the agent. The agent's computational processes manipulate symbols purely on the basis of their brute identity as arrays of ones and zeros. The agent undergoes a deterministic sequence of discrete states; a finite specification ultimately spells out how each state is extended, revised, or streamlined in its successor, based just on the readings of the agent's sensors and the state of its digital memory. Our correspondence between the agent's state and its environment does not affect how an agent acts. Nevertheless, symbolic programming is an essential tool in AI, because WE so much prefer to understand an agent's behavior in terms of the meaning of its representations. Symbolic programming allows us to draw on the results of computational logic and probability in agent design, and thereby to make the meaning of representations real. In realizing such designs, we derive agents that appear to act DIRECTLY based on the information they have about the environment—regardless of how that information is represented or computed. Such designs epitomize implementation in AI, not only for their intrinsic elegance but also for their natural connection with AI practices of modeling and evaluation, which establish and assess information about the environment.

2.3 Agents Must Be Engineered

AI is an engineering methodology, dedicated to understanding the empirical performance of agents as a guide to agent design. That means that after the modeling and implementation for an agent comes a thoroughgoing evaluation, spanning the concerns of AI research. For starters, we must determine how useful the agent proves for the overall task for which it was designed. In this assessment, nothing substitutes for observed runs of the agent, both in realistic field tests and across specifically controlled protocols of execution.

A successful design is not established merely with a satisfactory product; we must also evaluate our agent as a computational process. In particular, we can ask how well our algorithms perform not in idealized or worst-case conditions, but when running on specific hardware under a specific distribution of problem instances. Such questions become particularly important when an agent reasons from its model approximately rather than exactly. In such cases, the very behavior of the agent depends on the empirical properties of its computations.

Most abstractly, but equally importantly, we must undertake an evaluation of our models themselves; the construction of our agent enables new investigation of genuinely scientific questions. How well do the models that we have constructed to describe the environment of our agent actually fit the situations our agent encounters? Are the approximations and idealizations about the world that we have adopted in our model appropriate for practical problem-solving? At long last, is the world as we thought it was?

Evaluation stands on its own as a feature of AI practice, but it revolves around modeling every bit as much as does design. Whereas design involves models of the agent's environment, in evaluation, we model the ENSEMBLE of the agent coupled with its environment. Our evaluation model distinguishes a particular hypothesis about the running agent from a family of plausible alternative descriptions of it; each description maps out the distribution of behavior we should expect from a suite of events in which our agent runs.

Now, we are rarely fortunate enough in trials of our agent to observe exactly the behavior that our evaluation hypothesis predicts, and rarely unfortunate enough to observe results that completely diverge. The difference between what we expect in trials and what we see is a matter of degree. Modeling in evaluation serves as the background for the judgments we report in assessing this actual observed behavior. By modeling the agent in its environment, we can quantify how surprising experimental results are in light of a specific hypothesis about the running agent.

In evaluation, we can easily be proved wrong, but we will never discover that our agent works, that our computations are accurate, that our model fits. The best we can hope for is a provisional go-ahead—an invitation to attempt a richer model, a more complex computation, even just a more strenuous evaluation. No research practice rewards those who seek definitive answers. The appeal of AI is the ability to ask with absolute freedom, "what if?"—what does it mean to view the world, and ourselves, against a specific set of assumptions—and the ability to flesh out the question into a model, to realize the model in a real artifact, and set that thing loose into a world where, in the end, we can all judge "what if" for ourselves.

3 Decision Models: A Concrete Illustration of AI Practice

For the rest of this chapter, we will explore this general view of agent design. We start by considering a simplistic but general way of describing how an agent's observations and decisions can lead to successful or unsuccessful real-world outcomes. These descriptions are known as DECISION MODELS, and the process of constructing such models is known as DECISION ANALYSIS. Decision models do not originate within AI; decision models offer a perspicuous and principled tool for a wide range of fields, particularly business management, medical care and public policy, whose practice involves choice under uncertainty.[8] But here we will emphasize how computer scientists in particular can use decision models, as a mathematical approximation to the DESIGN SPACE for their agent implementations.

3.1 Modeling

To apply decision analysis to agent design, a designer starts by finding a decision that the agent will frequently need to make. This decision has to make sense to the designer, so the designer must identify the decision by finding a class of situations where the agent must act, and which deserve to be treated together. Moreover, since the agent has to realize the decision on its own, the designer should select a set of situations where the agent has a specifiable computational state. Finally, since the agent will apply the same decision analysis across all these situations, the situations should lead to the same distribution of outcomes for the same choices for the agent; for example, the agent's choices in different situations should not affect one another. In short, to apply decision analysis, the designer has to construct a CONTEXT under investigation, by finding a set of situations that the agent will enter in a distinctive computational state, and that the agent could face repeatedly without any repetition substantially affecting another. The problem for design is to determine what the agent will do in this context.

Observe that the context of investigation is characteristic of AI practice in setting up a link to the real world, and an approximation. When we link a distinctive state of the agent to a specific set of real-world situations, we help ensure that we have a meaningful subproblem for agent design, because we ensure that the computations we plan for the agent can and will determine the agent's responses to a clear range of circumstances. Indeed, it is common in robotics to call such a collection of coherent responses a BEHAVIOR, especially among those who emphasize the importance of equipping a successful agent with an appropriate repertoire of behaviors.[9] At the same time, in an open-ended world, we must expect that the agent's computational states across this set of real-world situations will be only approximately equivalent; we must expect that such states will arise not under just these situations exactly, but only approximately so; and we must admit that some of these situations exert some influence on some others, even if that influence can be ignored for practical purposes. We must be mindful of these approximations in implementing and evaluating our design.

Let us consider the implementation of a robot-pet consumer toy as an illustration. Its task, informally, will be to entertain its owners, by performing tricks such as fetching a ball thrown past it or dodging a ball rolled towards it. Our toy will have legs for locomotion, and a simple camera and touch receptors to sense its environment; we would doubtless want it to take the form suggestive of a familiar pet—a dog say.

A designer might devote special consideration to those situations where one of

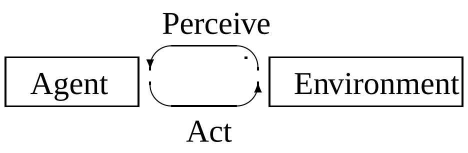


Figure 1: The agent's cycle of perception and action in its environment.

these tricks takes the dog from a brightly-illuminated area to a darker one. What if, for example, a slightly malicious child gets the dog to scoot through a doorway from a sunny room into a dark closet? The dog faces particular uncertainty and risks in such situations; the dog will have limited visual information until the aperture of its camera opens mechanically, and the camera can capture crisp images at the lower light level. Fortunately, since the dog can be provided with a trigger that identifies cases of insufficient illumination robustly (if never perfectly), the dog can be designed to handle these situations strategically.

Once we have a context of investigation, we can frame the design problem precisely. Our final agent will have some algorithm that it follows in this context; this algorithm continues until the agent's task is complete or until other conditions prevail and other algorithms take over. We suppose that the execution of this algorithm involves a sequence of possible states or CYCLES; each cycle offers the agent an opportunity first to PERCEIVE its environment and then to ACT to change it. Figure 1 offers a mnemonic depiction of the origin of this cycle of execution in the agent's interaction with its real-world environment.

In each cycle, the agent begins with expectations about its environment informed by its history, including the distinctive initial state of the algorithm, any sensor readings it has obtained during prior cycles of execution, and any past actions it has taken. The agent can then reconcile these expectations with the new readings about the environment available in this cycle. Finally, the agent selects an appropriate action. The algorithm we design will determine the agent's action at each stage as a function of the agent's history and sensor readings.

A decision model offers perhaps the simplest possible description of the space of such algorithms. A decision model represents the stages of execution of possible algorithms explicitly in terms of a history of observations and choices that the agent has made from its initial state. It represents the agent's expectations at each stage of execution statistically, in terms of the probability of the different observations that the agent may obtain at that stage. At any stage where the agent will act, the decision model lays out each of the possible actions that we might design the agent to take. Finally, for stages where the algorithm terminates, the model provides an assessment of the outcome that the agent has achieved with the complete alternative history.

The formal structure of a decision model is a TREE—that is, a mathematical object containing a set of NODES and a set of EDGES that lead from one node to another, where there is exactly one path from any one node to any other node. (A path follows a succession of distinct edges forwards or backwards.) Each node in a decision model represents a possible stage of computation for our agent, which the agent would reach in a subset of possible situations in the context of investigation. Each edge in a decision model represents a possible way that the computation could evolve in one step of perception or action.

Internal nodes in a decision model are of two kinds. The first kind, OBSERVATION NODES or CHANCE NODES, represent a stage of execution of the algorithm which obtains information from sensors. From the agent's point of view, these observations are unpredictable; the agent must be prepared for any possible result. At the same time, the agent may have some evidence about which results are more likely and which results are less likely.

Mathematically, we describe such an observation using a RANDOM VARIABLE associated with the observation node. This variable takes on one of a finite set of different VALUES, corresponding to the different possible results of the observation. Since the decision model must represent all possible alternative histories of computation, each possible value determines an edge to a new node, called a CHILD of the observation node; this child represents a further stage of execution in which the agent has observed this value. In addition, to formalize the evidence available to the agent about how its history meshes with the real-world situations of the context, each value gets a PROBABILITY between 0 and 1. Here, this probability is measured relative to the situations in this context which match the agent's history; the probability reports the fraction in which an agent would observe the specified value. Of course, the sum of the probabilities for alternative values of the variable must sum to 1.

Figure 2 illustrates the graphical convention that we will use to display observation nodes in decision models. The node is drawn with a circle with the random variable within; here it is X. (The use of a circular node provides a standard indication for a random variable.) The edges to the children of the node are labeled by value and probability; here we see $X = y$ with probability 0.8 and $X = n$ with probability 0.2. For our robot, we can understand X as a boolean observation, derived

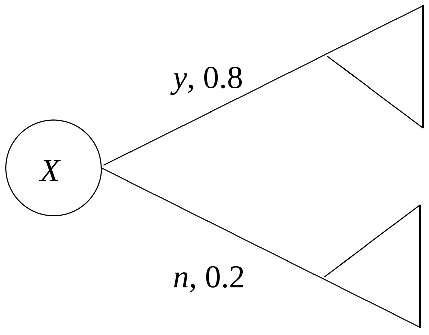


Figure 2: An observation node.

from automatic analysis of the latest image, which reports whether the agent's field of view contains a certain kind of blob (possibly indicative of a nearby toy ball and so possibly suitable for guiding a following behavior). Thus, the reading $X = y$ is evidence there is a ball to follow; the reading $X = n$ is evidence there is not. The submodels associated with these different outcomes are not explicitly represented; triangles mark where the subtrees have been suppressed for exposition.

The second kind of internal nodes, DECISION NODES or CHOICE NODES, represent a stage of execution of the algorithm at which an action can be performed. Our design space can include a range of different algorithms that involve alternative actions for each choice point. We assume that our design may freely select any of these actions at this stage of execution, whichever we think best. Accordingly, each decision node has a unique child for each alternative action that we are prepared to consider.

Figure 3 illustrates the graphical convention we will use for decision nodes. The node is drawn with a square; an index within the square, here 1, labels the choice for further discussion. (Again, a square provides a standard notation for a choice.) The label of an outgoing edges indicates the action taken at the decision node for that child's history. Here we consider three possible actions that might be available for our robotic dog: running a primitive following behavior, represented by f, to track a visual object; waiting for its cameras to accommodate to the ambient illumination, represented by w; and returning to home base by some robust behavior, represented by h (we might assume this behavior uses a different sensory modality from vision, such as a wireless beacon).

Finally, a leaf node in a decision model represents a state in which our algorithm has terminated and the agent has realized some distribution of possible outcomes.

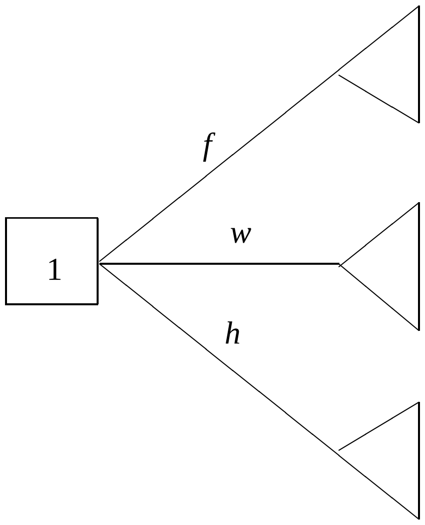


Figure 3: A decision node.

We will represent the degree of success that an agent achieves with a course of action in terms of a real number between 0 and 1, called a UTILITY VALUE. A utility value of 0 represents total failure; a value of 1 represents complete success. Intermediate utility values indicate partial success, the higher the better.

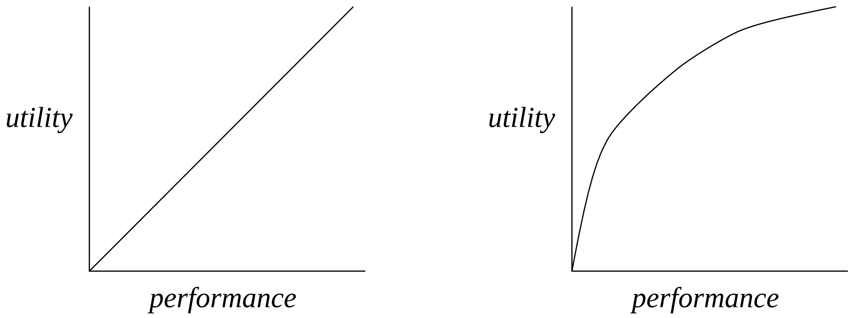
Utility is a human abstraction; it records both the preferences by which we as designers rank alternative real-world results, and our willingness to let our agent take risks whose results are only possibly favorable. As such, utility is not an arbitrary score but rather a precise tool, whose use reflects a specific collection of assumptions about what will make an agent successful. Similar assumptions govern any characterization of the outcome of a decision in terms of its utility.

To start, when we appeal to utility, we presume that we are explicitly committed in design to a specific measure of possible outcomes. In some decisions, the measure of outcome is easy: a small business might measure success by dollar profit. More generally, in agent design, many objectives are reasonable: to minimize the cost in dollars of the resources consumed by the agent; to maximize work achieved by the agent, as measured in the natural units of that work whether they be bytes transmitted in an embedded network or packages delivered in an augmented office; to maximize the satisfaction of the users or owners of the agents, as indicated by reported scores on a subjective rating scale; or several such objectives, in any weighted combination. Typically, our inspiration (or employment) in building an agent impels the choice. For our robotic dog, for example, we can expect performance to be measured by the reports that members of its target market give about how much they enjoy observing and interacting with the toy. To assess performance, we really would bring in users for trials with a prototype dog; after trial runs, users would rank the appeal of the dog on a numerical scale, perhaps using zero to indicate complete annoyance and five to indicate blissful entertainment. The higher the score, the better.

Our performance measure gives us information about specific situations that we will aim to achieve or to avoid. For example, we know that if one maneuver—fetching, say—always yields a user-satisfaction score of 4, then we should prefer it over another maneuver—rolling over, say—that always yields a score of 3. However, to accomplish our design, we will probably have to make decisions that are less clear-cut. For example, what if fetching sometimes succeeds and yields a score of 4 but often yields a score of 2, because the dog misses and can't recover? In this case, should we program the dog to fetch or to roll over?

The answer to such uncertain decisions depends on our attitude towards risk, which is independent of our performance measure. For example, suppose that we want our robotic dog to develop a long, engaging, and consistently satisfying relationship with its principal user, and we expect that an episode in which the dog upsets this person jeopardizes this objective in a way that one equally happy outcome cannot offset. Then, we would want the dog to prefer a strategy that leads to sure but mild satisfaction—rolling over, here—over another—fetching, say—that shoots for thrills but risks disaster. This represents what is known as a RISK-AVERSE strategy. Buying insurance offers a more familiar example of a risk-averse strategy. Buying insurance often makes sense for a small business: for the price of your insurance premiums, you can eliminate the cost you risk paying in an unlikely but bad situation. Of course, a small business should expect to pay more on average when it buys insurance than when it doesn't: after all, the insurance company expects on average to make a profit on that policy and others like it! This doesn't make it wrong to buy insurance. It just shows that even if a small business measures its profit in dollars, it should not necessarily make its decisions just by the dollars it expects to make. The performance measure doesn't capture its attitude towards risk.

Agents don't have to avoid risk. They can actively seek it out. For our robotic dog, we might expect that users would enjoy the variability and surprise of a risky strategy, and that a rare, outstanding success would maintain users' interest better than an accumulation of small, predictable wins. Then we would in fact want the dog to shoot for thrills and risk disaster: a RISK-PRONE strategy. Buying a lottery ticket is a risk-prone strategy from everyday life.



Utility for a risk-neutral agent. Utility for a risk-averse agent.

Figure 4: Performance, utility and risk. Graphs plot the utility of a sure outcome as a function of the performance attained.

Utility is a mathematical construction that combines a performance measure with an attitude towards risk. Utility agrees with the performance measure on outcomes that are certain—an outcome with higher performance also has higher utility. However, unlike natural performance measures, utility always extends to uncertain outcomes in an elegant way. The utility of a risky situation weights the utilities of the different possible outcomes in proportion to their probabilities. For example, we assess a risk that half the time pleases the user to utility 0.75 but is equally likely to please the user to utility 0.25 as worth 0.50; we regard this as an exact equivalent to a definite outcome that pleases users to utility 0.50.

It turns out that it is possible to construct a utility function that realizes any reasonable set of design preferences. To respect the performance measure, utility just has to be an increasing function of performance. Among such increasing functions, a suitable one is always available to represent our attitude towards risk. For example, making utility a linear function of performance gives a RISK-NEUTRAL strategy, suitable for design problems where many agent runs contribute independently to the overall benefit of deploying the agent or agents. Meanwhile, we can encode a preference for risk-averse strategies through a utility function that preferentially boosts lower values of the performance measure; we can encode a preference for risk-prone strategies through a utility function that discounts lower performance values. Figure 4 contrasts the relationship between performance and utility for risk-neutral and risk-averse agents.

Concretely, to get our dog to prefer to satisfy users less thoroughly but more of the time, we might transform a user rating of 2 to a utility of 0.5; a rating of 3, to a utility of 0.7; and a rating of 4, to a utility of 0.8. A 50-50 chance of outcomes of 2 and 4 determine a utility of 0.5 * 0.5 + 0.5 * 0.8 = 0.65. This fails to surpass the

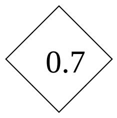


Figure 5: A reward node.

0.7 utility of a definite outcome of 3. In other words, the utility computed for 3 is boosted up to 0.7 over the rating of 0.65 you would expect from the utility computed for 2 and 4 by a linear model. Visually this is reflected in the concave downward shape of the risk-averse curve of Figure 4. When you draw a chord through the utility curve, you visualize all probabilistic combinations of the two endpoints of the chord (line segments are linear combinations). Horizontal coordinates along this chord indicate the average performance measure of the combination while vertical coordinates indicate its utility. Since these segments always lie below a concave downward utility curve, it shows that such models prefer the sure outcome, as plotted in the utility curve, over the corresponding uncertain outcome.

By comparison, to get our dog to take a chance on thrills, we could transform a user rating of 2 to a utility of 0.2; a rating of 3, to a utility of 0.4; and a rating of 4, to a utility of 0.8. Now a 50-50 chance of outcomes of 2 and 4 determine a utility of 0.5 * 0.2 + 0.5 * 0.8 = 0.5, which does surpass the definite outcome. Here the utility computed for 3 is penalized down to 0.4 over the linear rating of 0.5; now this would be reflected visually in a concave upward curve.

Figure 5 illustrates the graphical convention we will use for leaf or REWARD nodes. The diamond is the standard notation for an assessment of outcome; the number within reports the utility achieved.

In our discussion thus far, we have already presented the key ingredients of a decision model for our robotic dog. In summary, we have identified a context for agent design in which the robotic dog chases a ball into a darkly-lit region; the low illumination of the agent's cameras in these situations sets up a computational context for specialized decision-making. In this context, the agent has three actions at its disposal: to continue the chase however it can (f), to wait for its cameras to accommodate (w), and to return home (h). It can schedule any of these actions at any stage of execution. However, we rule out two consecutive steps of waiting, because once the agent has waited for its cameras to accommodate, further waiting will not improve the images it obtains but will make the agent less likely to catch the ball or otherwise entertain its owner. Meanwhile, as events develop, the agent has

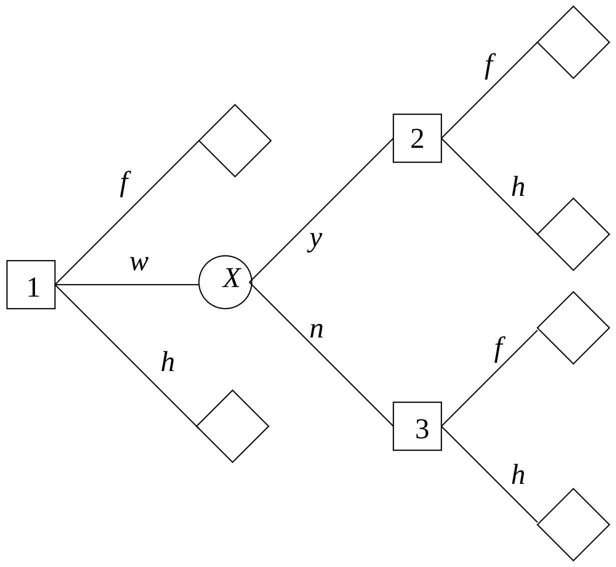


Figure 6: Design space for the robotic dog context.

access to a binary observation X. The reading $X = y$ provides excellent evidence that the agent sees the ball it was chasing; the reading $X = n$ provides excellent evidence that it does not.

These considerations determine a design space for the agent as illustrated in Figure 6. The agent begins at node 1 with an initial choice of action. If we design the agent to follow the ball or to return home, we immediately determine the agent's performance in this situation. If, however, we design the agent to wait, we offer the agent a second perception-action cycle; the agent's next action is conditional on its reading for X. At this stage, at decision points 2 and 3, we call for the agent to commit to follow or to return home. Consequently an outcome is assured.

Figure 6 describes our alternative designs, but does not describe the success that those designs will achieve in the actual circumstances in which the robot will act. For that, we must provide the probabilities and utilities that Figure 6 as yet omits.

In special cases, we will be able to supply these parameters as part of our specification of the agent. Typically these are cases where the design team can base any values they provide on substantial experience with the task context. Even with such expertise, specification is contraindicated when the exact parameter values greatly affect the design, either because small changes in those values can affect which actions the agent should take or because they can induce substantial differences in the utility the model predicts for those actions.

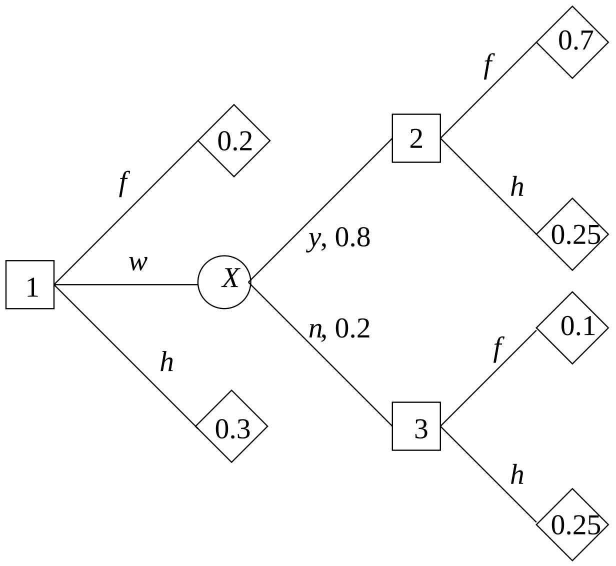


Figure 7: Full decision model for the robotic dog context.

The alternative is to determine parameter values by empirical investigation. For our robotic dog, that would mean building a prototype that could exhibit the full gamut of possible behavior for the context under investigation. We then estimate parameter values by collecting a range of possible histories with this prototype and assessing the outcome the prototype obtains in each. In other words, this is a problem of MACHINE LEARNING, a topic we return to in Section 4.1.

Let us assume that we have applied one or the other of these methods, and have obtained the results schematized in Figure 7. Specifically, we find that users find a successful chase quite entertaining (0.7). On the other hand, for the robot to chase after a ball it can't see is hardly any fun (0.2), and it is all the more boring when the robot waits before doing so (0.1). In between are cases where the robot just returns home, either immediately (0.3) or after some delay (0.25); while these outcomes are not themselves fun, they at least afford another game. Finally, as Figure 7 again shows, we find that our robot is able to reacquire the ball with probability 0.8 after waiting to get a better image.

By this point, you may be quite keen to know what it is that our robot should actually do. The answer is the algorithm—or as we shall soon say, the POLICY—presented informally in (1).

(1) Wait. Then, if $X = y$, follow; if $X = n$, return home.

Remembering that the utility of an uncertain outcome is the utility of its specific alternatives, weighted by their probability, you should be able to convince yourself that by implementing this policy, the robot can achieve a utility of 0.61.[10] You can also check that no alternative choice for our agent at any stage of execution will yield higher utility than what (1) provides at that stage.[11] In fact, then, our design space does not offer any policies that improve on (1).

3.2 Implementation

In building our robotic dog, or any other AI agent, we have the opportunity not only to implement specific policies such as (1) but also to work with classes of models and classes of policies in a general way. It is these general computations that offer the most compelling demonstration of the flexibility and elegance of physical symbol systems. This section offers an illustration: here we consider a general implementation of representation and reasoning with decision models.

In presenting this implementation, I will adopt the practice, common in computer science, of using PSEUDO-CODE, which abstracts from the details of any particular programming language and allows the reader to adapt the discussion easily to the programming language of their choice.[12]

3.2.1 Representation

We begin by describing representations for decision models and for policies. Our pseudo-code assumes basic types for INTEGERS, as typically realized as binary numerals of fixed length; REAL NUMBERS, as typically realized in accord with standards for floating-point arithmetic operations; and a further basic type, for SYMBOLS, which is rather distinctive to AI. The symbol type provides a set of arbitrary elements for which a programmer maintains an arbitrary and open-ended correspondence with constituents of a real-world environment; programmers specify symbols as character strings and the computational realization of symbols is typically closely related to these strings, but may streamline the implementation to speed up computations.

Our pseudo-code also assumes three ways of assembling elements together into larger representations, or DATA STRUCTURES. A LIST groups together any number of data elements of the same kind into a sequence whose elements can be accessed only by stepping forward one element at a time. To specify a list, I will give the elements in brackets, as $[1,2,3]$; when describing computations that apply uniformly across a list, I will also use ellipses, as $[a_1 \ldots a_n]$, and indices, as a_i. A TUPLE groups together a specified number of data elements of specified kinds into a structure each of whose elements can be accessed directly. To specify a tuple, I will give the elements in parentheses, as $(1,2,3)$. Finally, a NAMED STRUCTURE provides a way of grouping together heterogeneous collections of data when a representation takes qualitatively different forms in different cases. Each instance of a named structure is written fT where f is a symbol that identifies which form the data takes and T is a tuple that specifies the specific data elements for this instance; the definition of the named structure describes which such cases are possible.

Thanks to these assumptions, we can specify decision models using named structures as in (2).

(2) A MODEL is a named structure consisting of one of the following:

- $obsv(X, os)$ where X is a symbol (representing the observation being made) and os is a nonempty list of tuples (v_i, p_i, m_i) with v_i a symbol (representing an observed value), p_i a real (representing its probability), and m_i a model data structure (representing what happens next);
- $dcsn(n, as)$ where n is an integer (indexing the choice) and as is a nonempty list of tuples (a_i, m_i) with a_i a symbol (representing an action to take) and m_i a model data structure (representing what happens next);
- $rwd(u)$ where u is a real (representing an attained utility).

The three cases correspond to the three kinds of nodes introduced earlier. The symbol *obsv* keys an observation node; the parameter X represents a variable that the agent can observe, and the list *os* represents both the observations that the agent might make, and the consequences of those observations. The symbol *dcsn* keys a decision node; n indexes a specific possible choice for the agent, and the list *as* determines both what the actions the agent can take and what consequences those actions have. The symbol *rwd* keys a reward node; the parameter u represents the utility of outcomes that the agent has achieved. Note that the hierarchical structure of decision models is realized by the recursion in the definition of (2), which allows *obsv* and *dcsn* structures to contain other model structures as constituents. This recursion grounds out in *rwd* nodes as the base case.

Figure 8 shows how we write the model of Figure 7 as a data structure according to the conventions culminating in (2). Figure 8 doubtless helps motivate the graphical format of Figure 7 as a means for people to communicate decision models! At

```
dcsn(1, [(f, rwd(0.2)),
        (w, obsv(X,[(y,0.8,dcsn(2,[(f,rwd(0.7)), (h,rwd(0.25))])),
                    (n,0.2,dcsn(3,[(f,rwd(0.1)), (h,rwd(0.25))]))])),
        (h, rwd(0.3))])
```

Figure 8: Data structure for the model of Figure 7.

the same time, however, it underscores that any depicted model also corresponds to a concrete data structure that can be recreated on a machine.

By the same token, the interpretation we sketched for graphical decision models in Section 3.1 now allows us to interpret any model data structure as embodying a precise claim about the world. We start from our design context. This context gives us a correspondence between the action symbols in a data structure and the real actions our agent can take; it gives us a correspondence between observation symbols and the real computations our agent could do to obtain a reading from a specific sensor; and it gives us a correspondence between value symbols and the real readings that our agent could derive from its sensors. Moreover, the context determines the distribution of real-world situations which a model structure as a whole must describe, for the model proceeds from a computational state triggered in just these situations and spells out the possible consequences of the agent's subsequent actions.

In general, then, each substructure of the whole model describes the result of particular observations and actions in the context and so will make a CLAIM about a corresponding subset of these situations.

Drawing on this background, we can formalize such claims as in (3). The definition, with its basis clause (3a) and inductive clauses (3b) and (3c), mirrors the recursive composition of model structures.

(3) a A structure $rwd(u)$ claims about situations S that these situations give the agent an expected utility u, and no opportunity for further action.

b A structure $obsv(X,[(v_1,p_1,t_1)\ldots(v_n,p_n,t_n)])$ claims about situations S that these situations all provide the agent a reading corresponding to observation symbol X; that each result v_i occurs with probability p_i throughout S, determining corresponding situations S_i; and that the claims of each substructure t_i hold about situations S_i.

c A structure $dcsn(n,[(a_1,t_1)\ldots(a_n,t_n)])$ claims about situations S that these situations all provide the agent an opportunity to act in which the agent can be designed to do any of the actions corresponding to $a_1 \ldots a_n$, determining corresponding situations $S_1 \ldots S_n$; and that the claims of each structure t_i hold about situations S_i.

A body of definitions such as (3) that relates an agent's data structures recursively to conditions on its environment is called a COMPOSITIONAL SEMANTICS. A semantics allows us to interpret an agent's data structures as representations of its environment. The payoff for this interpretation comes in the operations that we define on the agent's representations. We will be able to show that these operations allow the agent to act as it should if its environment matches what it represents.

To proceed, we also need a data structure that can represent an algorithm by which the agent acts in its environment. As a computer scientist, one learns to take this very deep step quite casually—we can design data structures that represent anything we want, so we can even design data structures that represent rules for carrying out computations! (4) defines a named structure for such an algorithm.

(4) A POLICY is a named structure consisting of one of the following:

- $switch(X, ps)$ where X is a symbol and ps is a nonempty list of tuples (v_i, π_i) where v_i is a symbol and π_i is a policy data structure;
- $do(a, \pi)$ where a is a symbol and π is a policy data structure;
- a case *end* for which no further data is required.

Informally, $switch(X, [(v_1, \pi_1) \ldots (v_n, \pi_n)])$ represents a conditional algorithm in which the agent first carries out the observation corresponding to X and then continues with the policy π_i for whichever reading v_i is obtained; $do(a, \pi)$ represents a sequential algorithm where the agent first performs the action corresponding to a and then continues with the policy π; *end* represents the final stage of execution at which the policy ends. Thus the policy described in (1) can be implemented in the policy structure of (5).

(5) $do(w, switch(X, [(y, do(f, end)), (n, do(h, end))]))$

As with Figure 8, this example serves as a reminder that we could at any point recast our human descriptions in an explicitly computational formalism (and as an illustration of why we may prefer not to).

A given model represents an environment in which only select policies are guaranteed to be executable; we will call such policies FEASIBLE in the model.

(6) a In a model $rwd(u)$, *end* is the one and only policy that is feasible.

b In a model $obsv(X,[(v_1,p_1,t_1)\dots(v_n,p_1,t_n)])$, a policy is feasible if and only if it takes the form $switch(X,[(v_1,\pi_1)\dots(v_n,\pi_n)])$, where each π_i is feasible in t_i.

c In a model $dcsn(n,[(a_1,t_1)\dots(a_n,t_n)])$, a policy is feasible if and only if it takes the form $do(a_i,\pi)$, where π is feasible in t_i.

(6) links models recursively to policies with a corresponding structure—(6a) provides the base case, for both models and policies; (6b) and (6c) describe inductive cases that simultaneously characterize complex models and complex policies.

A model determines the expected utility of any policy that is feasible in it, by describing the probability of alternative outcomes under the policy and the utility associated with each outcome. The overall utility (as always) sums the utility of each outcome weighted by its probability. You should be able to construct a recursive definition by cases, analogous to that in (6), which calculates this utility.[13]

3.2.2 Reasoning

Now we turn to the main problem: given a decision model t, compute a feasible policy that has the highest expected utility in t. Such a policy is called an OPTIMAL policy for t. The significance of this problem derives from the compositional semantics we have provided for decision models. Suppose that t is an accurate model, so that the claim associated with t truly describes our agent's environment. Then when the agent acts in this environment, it must carry out a policy that is feasible in t. Moreover, each feasible policy actually has the utility predicted for it by t. Thus, when t is an accurate model, an optimal policy obtains results for our agent that cannot be improved. This section describes a procedure OPTIMUM that solves this key problem, and justifies the details of this procedure in a mathematical way.

In order to streamline the mathematical development, we will avail ourselves of the certain constructs in our pseudo-code for procedures. First, we specify procedures by cases, depending on the form of representation of the available data. For example, for OPTIMUM, the data is the model structure t; t may take the form of an observation, a decision, or a reward, and in each of these cases a different course of action is indicated. Be aware that we are abstracting away from the alternative mechanisms programming languages have to achieve such definitions: some do it directly, in procedure definitions; some do it indirectly, by packaging data together with special-case METHODS of operation into OBJECTS; others can only do it explicitly, with conditional statements.

Second, we assume that procedures can construct an arbitrary structure of data, and RETURN it as a result. For OPTIMUM, the result will be a tuple (π, u) reporting both the optimal policy π for t and the maximum utility u that the agent can achieve in t. (Evidently, u must be the utility of π in t.) Again, you would do well to remember that programming languages do not all make it equally convenient to deliver such complex results.

Third, we will use general assignment statements of the form in (7) to indicate that the values in the specified *structure* are determined through carrying out the specified *computation*.

(7) $\quad$ *structure* $\leftarrow$ *computation*

Our definitions will assign a value to any variable at most once. The convention of assignment may therefore be regarded as an instruction to reuse a computed value (like a kind of abbreviation); it does not require the use of program variables with mutable values, which can be changed repeatedly over the course of a computation.

Finally, we will use mathematical notation to specify such routine calculations as summation, finding the largest element in a list, applying a common operation to all the elements of a list, or constructing a new list in a uniform way.

Pause, then, to consider the procedure OPTIMUM as defined in the three cases of (8).

```
(8) a   OPTIMUM $rwd(u)$
            RETURN $(end, u)$
    b   OPTIMUM $obsv(X, [(v_1, p_1, t_1) \dots (v_n, p_n, t_n)])$
            for each $i$: $(\pi_i, u_i) \leftarrow$ OPTIMUM$(t_i)$
            RETURN $(switch(X, [(v_1, \pi_1) \dots (v_n, \pi_n)]), \sum_i p_i * u_i)$
    c   OPTIMUM $dcsn(n, [(a_1, t_1) \dots (a_n, t_n)])$
            for each $i$: $(\pi_i, u_i) \leftarrow$ OPTIMUM$(t_i)$
            $d \leftarrow$ any $i$ that maximizes $u_i$
            RETURN $(do(a_d, \pi_d), u_d)$
```

We wish to show the CORRECTNESS of this definition; that is, we wish to show, for any tree t, that OPTIMUM t returns a pair with the optimal policy for t and the maximum utility possible in t. In keeping with the form of Definition (8), which sets out the result of OPTIMUM on a larger model in terms the result of OPTIMUM on smaller model, we will argue for correctness by STRUCTURAL INDUCTION.

Each model structure has a maximum length in the number of steps of perception and action that it offers the agent; call this the HEIGHT of the model. In structural induction, we group together models by height, and consider what happens as height gradually increases.

The base case comes in a model with height 0, which must take the form $rwd(u)$. By (8a), here OPTIMUM returns (end, u). This is the only feasible policy, so it must be optimal, and of course u is its utility.

We now hypothesize for the purposes of argument that OPTIMUM is correct for models of height h or less, and consider a model t of height $h+1$, for which OPTIMUM returns (π, u). This model must be constructed inductively from smaller models, as an *obsv* or *dcsn* structure. Now u will be the utility of π, as you can easily verify,[14] so it suffices to show that the utility in t of any feasible policy π' is at most u. We prove this by considering separately the two possible forms that t could take: $obsv(X, [(v_1, p_1, t_1) \dots (v_n, p_n, t_n)])$, where (8b) applies; and $dcsn(n, [(a_1, t_1) \dots (a_n, t_n)])$, where (8c) applies.

In the first case, π and π' are conditional policies. The overall model t includes a submodel t_i of how the world evolves with any particular reading v_i. In t_i, π directs the agent to follow a policy π_i, while π' directs the agent to follow a policy π'_i. Because t_i has height at most h, π_i is optimal for t_i by hypothesis; so the utility u'_i of π'_i in t_i does not exceed the utility u_i of π_i in t_i. Now, the utility of π, $u = \sum p_i * u_i$, while the utility of π', $u' = \sum p_i * u'_i$. Since in each case $u'_i \leq u_i$, $u' \leq u$. But π' is arbitrary: it follows that π is optimal for t.

In the second case, π and π' are sequential policies. The overall model t includes a submodel t_i of how the world evolves with any particular action a_i. According to the clause (8c) which computes π, π is constructed first to perform that action a_d for which the OPTIMUM algorithm for submodel t_d leads to a policy π_d with the best results. (Of course, π continues with π_d.) Meanwhile, π' spells out some initial action a_i and a policy π'_i to follow afterward. Since the corresponding submodel t_i of t has height at most h, OPTIMUM computes an optimal policy for t_i with some utility u_i. The utility of π'_i and thus the utility of π' cannot exceed u_i. At the same time, in (8c), we chose π_d, based on its utility, when the optimum policy π_i for t_i and its utility u_i was a possibility; that means that π_d and thus π must have utility at least u_i. Again the utility of π must be at least that of π' and since π' is arbitrary it follows that π is optimal for t.

We have thus established that OPTIMUM starts out treating models correctly, and that as models increase in height arbitrarily, OPTIMUM continues to treat them correctly. Thus, we have established that OPTIMUM is correct for all models.

I hope that the rigor of this discussion, which epitomizes the precision possible in studying computational operations on representations, has not obscured the central insight which lies behind the procedure of (8) (and its correctness proof, too). To decide how to act, an agent looks forward into the future as far as it can. In the case of decision models, that limit is set by reward nodes, which offer the agent no further actions. From there, the agent works backwards towards the present; at each step it gradually works out the best policy to guide its action in whatever decisions await, but at the same time eliminates from consideration a vast array of policies that will not fare as well. In this way, as the agent goes from step to step, it can focus its deliberation on just its good, live options.

The OPTIMUM procedure is just one of many useful algorithms for working with decision models and policies. An obvious further example is the procedure to EXECUTE a specified policy. Such a procedure must presume a primitive GET that obtains the reading of a specified variable, and a primitive RUN that carries out a specified action. With these resources, the procedure can be specified recursively by case analysis on the structure of policies, along similar lines to OPTIMUM—as you are invited to verify yourself.[15]

3.3 Evaluation

Let's survey the steps outlined thus far. We have framed our problem, by looking to the world for a coherent task and considering diverse approaches to solve it. We have refined this design space into a model of the agent's environment and task, by assessing the performance of an initial prototype. And we have drawn on general procedures to create a decision-making strategy for our agent that best fits this model of the world. We have, in short, done our very best to make sure that our agent is going to work. And why should we not have? Careful development may be painful, but real-world failure is worse.

Despite these efforts, however, success is never assured. Each step has its dangers. Basic approximations and design assumptions may not suit the open-ended environment in which the agent must act. Training data, too, may misrepresent the actual environment, whether through changing circumstances or just bad luck. And a real implementation is not an idealized mathematical construction but a human and imperfect creation, in which frank errors cannot be ruled out.

It is fair to ask, then, does the system work? Does it in fact work for reasons we understand? How might such agents be improved in the future? Any answers to such questions are ultimately subjective judgments, but the premise of evaluation is that they are judgments for which general guidelines are available.

The judgment of whether we understand the actual performance of a final agent (the second of our three questions) is often the clearest. We can base this judgment directly on a model of our agent's PERFORMANCE in its environment. A performance model describes the full distribution of outcomes that the agent will achieve by following its policy. As such, it may naturally extend the basic model of the environment that we use to construct the agent.

Take decision models. A decision model is an incomplete description of the distribution of an agent's outcomes in its environment, because each leaf in the model summarizes a whole distribution of outcomes by a single utility value. The model of Figure 7, for example, just says that on any trial our agent has a 0.8 probability of achieving outcomes that have utility 0.7 ON AVERAGE and a 0.2 probability of achieving outcomes that have utility 0.25 ON AVERAGE. This is useful for the agent: the agent really is indifferent between a certain outcome with utility 0.7 (say), and two equally likely outcomes, one with utility 1.0 and one with utility 0.4. But the difference matters for us for evaluation: in the first case, a trial where the agent sees an outcome of 0.4 may be quite unexpected; in the second case, it is routine.

In the case of decision models, then, the performance model augments the world model by information about the distribution of possible utilities at each leaf. Naturally, this distribution can be estimated from the outcomes in training, much as the expected utility itself can be. (But you must plan for the evaluation!) In turn, the separate distributions determine the overall distribution of outcomes for a particular policy.

In practice, we just need to estimate the overall variability of the outcomes under a policy. We can measure this by a statistic called the VARIANCE of the distribution of the observed utility u. If the mean value for u is μ (here the expected utility), the variance σ_u^2 is the value we expect for $(u-\mu)^2$. Let's assume that we have found that the observed utilities for different outcomes, for the model of Figure 7, cluster closely around their average values; in this case we could have a value of σ_u^2 of perhaps 0.036.

In order to use this model to assess the performance of the agent, we perform a substantial number n of representative runs with our agent and tabulate the results. Now, individual trials may have unpredictable results, but, if our model is good, our agent's average performance u_e across all n trials of an experiment is very likely to come close to the model average μ. Any individual data point can have an extreme value, but for the average performance u_e to have an extreme value, a large proportion of the trials from the experiment must happen to diverge from μ toward

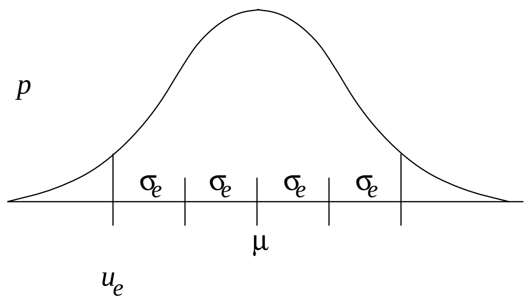


Figure 9: Idealized histogram showing relative probability of different experimental results u_e for average utility across n trials.

the same extreme.

Figure 9 depicts the situation mathematically. It envisages an idealized histogram of the results that our model predicts we would obtain, across many experiments with n trials. The peak in the histogram occurs when the experiment sees outcomes whose average utility exactly matches the model mean μ. As the average observed utility diverges from the model mean μ, results become increasingly rare, although any result remains possible in principle. The narrowness of the peak reflects the variance σ_e^2 of results across multiple potential experiments. We expect this variance to decrease as more trials are run in the experiment. In fact, it can be shown under reasonable assumptions that σ_e^2 is approximately equal to σ_u^2/n, so as n increases the experimental results are more precise; with probability approximately 0.95 the result of an experiment u_e will lie no further than twice σ_e from μ. (Note that by repeating the experiment, we do not change our expected average, namely μ. We simply reduce σ_e, the "error" we expect between the experimental result and μ.)

We can now regard our experimental trials as a test of the HYPOTHESIS that the performance of our agent in the real world matches our model of its performance. Our experimental trials have an average utility which diverges by some amount d from μ. A divergence as large as d or larger could always be due to chance. Figure 9 illustrates how our performance model quantifies the probability of such chance divergences. In particular, if $d > 2\sigma_e$, the odds of seeing such a divergence by accident are worse than one in twenty. Such unlikely results should raise our suspicions about the model.

In forming a judgment about our hypothesis, we must consider not only the fit between the hypothesis and the data but also the alternative hypotheses are available. Suppose that our results are suspicious. If no available hypothesis explains our results better, our results do not speak against our hypothesis; we simply know that our results truly are unlucky. On the other hand, when competing hypotheses about our agent's performance would fit our observed results better, the poor fit of our hypothesis to the data does provide evidence against our hypothesis.

A standard measure that reconciles these two considerations is the risk of being wrong if our performance model is correct, but we judge our experimental results to be in conflict with it. We calculate this by considering the probability of obtaining results that diverge from our prediction as much as those we have obtained and that some alternative model explains better. For most scientific purposes, we are willing to accept a risk of error of one in twenty. In most agent performance evaluation, we anticipate that our agent could achieve any utility as an alternative; the interactions we do not model could help or hurt the agent. Accordingly, we count an experiment as calling a performance model into question when the observed divergence d is greater than roughly twice the divergence σ_e expected in the experiment.

As a concrete illustration, suppose we run an experiment in which our robotic dog performs the accommodation-and-chasing procedure developed in Sections 3.1 and 3.2 40 times. In these trials, it achieves an average utility of 0.65. By eye it looks as though our agent is doing very well—almost too well. Is it? With this experiment, $\sigma_e^2 = \sigma_u^2/n = 0.036/40 = 0.0009$; so $\sigma_e = 0.03$. Thus, we reject the model only if the observed divergence exceeds 0.06. Here the divergence is 0.04. It appears that our experimental result of 0.65 is not so unexpected.

We can evaluate other claims about our agent's performance along similar lines. For example, we often want to report that our agent is a success. Typically, we mean by this that our agent yields an improvement over what was previously possible, and we have to test the alternative that our agent performs the same as others, or worse.

Suppose that, in addition to experimental results from our own agent, we have experimental results from a benchmark agent, solving real-world tasks under identical conditions. For new problems, the comparison should be with a simple but reasonable agent; thus our robot dog suggests a benchmark design which always just returns home in cases of difficulty. For well-studied problems, of course, other researchers' work supplies the benchmark agents, and it may not even be necessary to obtain one's own experimental results for them. Agents' performance is often published for standard sets of training and test data made available by the research community.

We will have observed some improvement over the benchmark in our experiments with our agent; otherwise the experiments certainly provide no evidence that our agent is an improvement! As part of our judgment of evaluation, we have to pronounce our opinion as to whether the observed difference reflects a genuine fact about the agents' real-world performance, or whether the difference might have originated by chance. Here the key step is to devise a mathematical argument about the probability of obtaining an improvement as large as we did by chance, if our agent and the benchmark agent really have the same performance. It obviously bolsters our case if this probability is very small.

Where evaluation against a basic benchmark provides a check for the progress we have made with an agent, evaluation against a suitable upper bound gives insight into how much room our design leaves for improvement. Such bounds may be obtained experimentally by investigating how well expert humans are able to perform in the agent's task, or by investigating alternative agent designs whose training and test data sets include important task information that would not normally be available to an agent in practice. More generally, with suitable experimentation and comparison (always backed up by statistical argument), it becomes possible to defend precise claims about the situations, the representations, or the decisions that underlie an agent's strengths or contribute to its weaknesses. Such judgments can be an invaluable guide to further research.

4 Decision Models and Paradigmatic AI Problems

Decision analysis not only epitomizes the practice of agent design, but also manifests the enduring challenges of AI research. In this section, I use decision models as a starting point to delineate these directions for investigation. On the one hand, decision models illustrate all the general difficulties inherent in connecting representations with a complex and uncertain world. Chief among these is the problem of MACHINE LEARNING, exploiting an agent's experience in its environment to ground its models and improve its performance.

On the other hand, decision models turn out to be badly unsuited for most real-world tasks. This leads to a search for new models. The search responds not only to the diverse applications of AI—areas such as ROBOTICS and COMPUTER VISION; HUMAN-COMPUTER INTERACTION and NATURAL LANGUAGE DIALOGUE; MEDICAL and SCIENTIFIC INFORMATICS—but more generally to the diverse cognitive abilities which people manifest across different domains. Better models capture insights that we discover about the real world, in representations that are more general, more robust, or more concise than decision models. But new models also bring new kinds of problems; a case in point is PATTERN RECOGNITION, the computational investigation of discrete responses to a continuous environment.

4.1 Problems of Learning and the Complexity of Models

In accounting for the limits of decision analysis—or any very flexible class of real-world models—the central concept is the COMPLEXITY of the model as a description of an agent's environment. A model is complex to the extent that it is free to represent arbitrary relationships that might hold in the environment. By this definition, we may argue that decision models are in fact as complex any model could be.

To see why, suppose we have a decision model t and a policy π that is feasible in t. For expository purposes, we will assume that all executions of policy π terminate after exactly $n+1$ steps of action, and we will assume that each cycle of perception feeds the agent the value of a binary observation variable. We label these variables X_i with i ranging from 1 through n.

Familiar arguments show that the model t assigns 2^n possible outcomes to policy π.[16] This structure is EXPLOSIVE in size, in the sense that small increases in the number of steps required to complete the agent's task will lead to enormous increases in the computational resources required to represent and reason about the agent's progress. For real-world action, however, it is important to emphasize another side to this explosion. Small increases in the number of steps required to complete the agent's task will lead to enormous increases in the amount of REAL-WORLD EXPERIENCE required to construct a full and meaningful model.

The explosion originates in the fact that the model t and policy π are capable of determining an ARBITRARY joint distribution on the values of variables X_1 through X_n. To make this precise, let $P(X_1 = x_1 \ldots X_n = x_n)$ denote the probability that the observations X_1 through X_n take on a specified combination of readings x_1 through x_n during a history with the agent. Then a joint distribution is any assignment of probability values for $P(X_1 = x_1 \ldots X_n = x_n)$ across possible observations, and whatever this assignment may be, we can construct a decision model to realize it.[17]

Now, in Section 3.1, we saw that, except in those few cases where designers bring very precise and accurate background knowledge to the construction of a model, the free parameters of a model must be estimated from a set of training data. The field of MACHINE LEARNING[18] offers diverse frameworks for this estimation problem. A simple statistical approach is to look for a model that maximizes the overall likelihood of the observed data; a more sophisticated one is to interpret the data as giving indirect evidence about a restricted set of likely parameter values that we supply in advance. Still other approaches recover parameters from data by more general optimization techniques that have proven effective throughout a range of practical distributions of problem instances; popular instances of this kind of strategy include SUPPORT VECTOR MACHINES[19] and (despite their name) NEURAL NETWORKS.[20]

Machine learning also affords numerous perspectives on how to process the evidence derived from data. The data may come in a single batch; in this case, learning algorithms can use any data at any stage of computation and can report just the final set of parameter values. By contrast, in INCREMENTAL LEARNING, the learning algorithm iteratively reconciles a provisional model with a new observation to derive a better model. Moreover, when action is involved, we may emphasize how the agent's policy changes with new observations (and refinements to parameter values); this is the perspective of REINFORCEMENT LEARNING.[21]

For all this diversity, no machine learning method can overcome the fact that training data is noisy. In general, empirical observations of the real world reflect underlying regularities of the world only partially. Our introduction to evaluation illustrated this point already in detail. We saw that repeated observations are necessary in order to derive an estimate for a parameter value that will be close to the true value with high probability.

Thus, to build a decision model from data requires repeated observations of each of the 2^n possible outcomes. With a training set of practical size (and don't forget that the limit is often the real-world effort—even the real-world cash—required to obtain the training data) and with a decision model of substantial complexity, it is inevitable that many of these outcomes will not occur, or will occur so infrequently that any derived parameter estimates are quite likely to be inaccurate. Under such circumstances, decision models are unlikely to be useful.

Sensor readings with a large space of possible values provide a complementary illustration of the complexity of decision analysis. Consider an observation D that approximates the distance to some obstacle. In a digital computer, the values for D will be discrete; the number of different possible values will be finite. In principle, a decision model can describe this, by inducing a distinct submodel for each alternative reading. When the number of alternative values for D reaches 256, 65536, or more, the amount of training data required also skyrockets; the model again becomes hopeless. An effective model will have to treat D as a continuous variable, and agents will have to learn the model and reason from it to determine the ranges of values for D which call for different responses.

4.2 Problems of Modeling and the Complexity of Environments

We see how severe a drawback this complexity of decision models can be, once we distinguish between the complexity of an agent's models and the complexity inherent in the agent's environment. An ENVIRONMENT is COMPLEX to the extent that its outcomes inherently reflect intricate interrelationships among many causal processes, which have to be discovered empirically. A complex environment offers little basis to generalize from one circumstance to another; in a complex environment, designers may have no alternative but to equip their agents with a model that is accurate and that therefore is complex, too. In a complex environment, designers must come to terms with the concomitant difficulties in computation and training.

However, a wide range of important environments must be simple, in the sense that important interactions can be described by a small number of empirical parameters. Indeed, a computational perspective on human intelligence suggests that all the behaviors that people naturally develop appeal to simple models of the world. There wouldn't be time to learn descriptively complex models of a causally complex environment from the limited experience people get. Readers familiar with linguistics or psychology will recognize this as a version of the Chomsky's POVERTY-OF-THE-STIMULUS ARGUMENT: an intelligent agent's open-ended behaviors must be informed not only by limited experience in the world but also by innately constrained models of the world.[22] In this connection, it is worth reinforcing that the inference to restricted models of the world applies regardless of how learning is understood or how learning proceeds—the argument depends only on the intractability of optimizing large numbers of parameters simultaneously.

A good way to think about practical models in AI and cognitive science is in terms of the REAL-WORLD CONSTRAINTS that a model adopts. These real-world constraints are statistical assumptions about the relationships among variables that allow agents to generalize from experience by rule. The advantage of such constraints is that they enable an agent to learn more quickly, by reducing the amount of data required to fit a model to the agent's environment. The challenge of such constraints is that the assumed relationships among variables must be borne out in the environment for an agent's learning to be effective. Typically, it is only possible to state meaningful constraints in a richly-structured representation of the environment, which makes causal and functional relationships explicit.

For example, consider designing a behavior where our agent classifies and responds to a new object in its environment. Think of our robot pet reacting to an approaching object that is brought to its attention by the looming in its visual field: Is this a toy, which the agent might pick up or fetch? Is this (part of) a person, who the agent might attempt to interact with? Or is this something dangerous, which the agent must avoid? In choosing its response to the object, the agent's performance depends on whether it deals with the object as befits its category.

Unfortunately, the agent cannot directly observe this. The agent has only its observations—some perceptual features of the new object. Our robot pet, for example, might be able to estimate an approaching object's speed, size, or shape. A decision model could only attempt to connect these observations directly to the choice of action the agent must make.

However, a model that incorporates an explicit representation of the true but hidden state of the world may be simpler, if it concisely describes the underlying generalizations in the environment. In our example, such a model could spell out how the agent's observations of an object correlate with the object's category. Toys might tend to be small, and move at a medium speed; people might prove slower, larger and characteristically blobby; dangerous objects might appear fast, or large, or even just pointy. By describing the baseline frequency of these different cases—say toys and people are relatively common while danger is thankfully rare—the model allows the agent to interpret its observations as giving evidence about what category that the object belongs to, and to react appropriately. In effect, then, this model links an agent's behavior to its CATEGORIZATION of its environment. Of course, in designing agents, we need not always think of objects or categorization as concretely as this; we can apply these models, and our intuitions about categorization, in open-ended ways.

To bring home the generality of this idea, let us consider a class of models that represents the hidden state that the agent must infer and respond to as a random variable S. The model specifies a constrained range of possible values for S, each of which corresponds to a distinct set of real-world circumstances that arise with a specified frequency in the agent's experience. Mathematically, the model represents this by a PRIOR distribution $P_m(S = s)$ on the alternative values for S.

The model also describes these circumstances (approximately), by associating each value of S with a characteristic distribution of observations and outcomes for the agent. The model represents the agent's observations as noisy, incomplete but independent clues to this state. Mathematically, for each observation variable O_i, the model specifies the LIKELIHOOD of seeing a particular reading o assuming the state has value s, or $P_m(O_i = o|S = s)$. Meanwhile, the model links the utility of each action just to the underlying state of the world. Mathematically, the model describes these outcomes with a table $U(s,a)$ of the utility expected when the state of the environment is s and the agent's choice of action is a.

Diagrams such as that in Figure 10 use the standard convention of a directed arrow between nodes to depict these probabilistic dependencies. In Figure 10, the state is represented by the random variable S. The agent's observations, represented

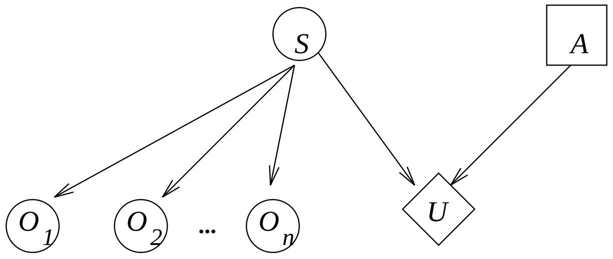


Figure 10: Naive Bayes Model

by random variables O_1 through O_n, appear as nodes linked only from S. The agent's utility, represented by the reward node U, is linked both from S and from the choice A that the agent makes. These representations amount to assumptions about the environment that we make in using such models to design an agent. When these assumptions fit, they allow an agent to reason successfully, to learn efficiently from limited training data, and to generalize reliably from it.

The agent's behavior involves making a series of observations and then responding with an appropriate action; the model provides the agent with grounds for this choice as follows. The agent first estimates the probabilities of alternative states of the environment, by matching its observations against the model. The agent then determines the most promising action, by calculating utilities in the model based on this inferred distribution. If the model describes the environment accurately, this policy allows the agent to act as successfully as possible.

More precisely, suppose that the agent has observed values o_1 through o_n for the variables O_1 through O_n. The model determines a probability distribution

(9) $$P_m(S = s | O_1 = o_1 \ldots O_n = o_n)$$

describing how probable any state of the world is given the observations the agent has made. Theis probability, called the POSTERIOR, represents the agent's BELIEF that the environment is in state s; thus the posterior distribution summarizes the information that the agent has about its environment. Using the posterior, the agent can determine the expected utility of any action a given its observations, which we write $U_m(a|O_1 = o_1 \ldots O_n = o_n)$, by weighting the outcome for a in alternative possible states s by the agent's current belief in s. (10) formalizes the relationship.

(10) $$U_m(a|O_1 = o_1 \ldots O_n = o_n) = \sum_j U_m(s_j, a) P_m(S = s_j | O_1 = o_1 \ldots O_n = o_n)$$

Of course, the agent's best strategy is to take the action with the highest expected utility.

The models depicted in Figure 10 are known as naive Bayes models. The Bayes part is because we use Bayes's Theorem[23] to calculate (10) from the parameters of the model. (11) shows the instance of Bayes's Theorem we need.

$$(11) \qquad P_m(S=s|O_1=o_1\dots O_n=o_n) = \frac{P_m(O_1=o_1\dots O_n=o_n|S=s)P_m(S=s)}{P_m(O_1=o_1\dots O_n=o_n)}$$

In words, Bayes's Theorem states that our BELIEF, that the environment is in state *s* given our observations, trades off the LIKELIHOOD of obtaining our observations if the environment was in state *s* against the PRIOR probability that the environment is in state *s*. We obtain a probability by normalizing for the overall probability of making our observations—the normalization factor can be calculated by summing the probability of being in a specified state and making our observations, over all possible states.

The model's "naive" independence assumptions entail (12).

$$(12) \qquad P_m(O_1=o_1\dots O_n=o_n|S=s) = \prod_{i=1}^{n} P_m(O_i=o_i|S=s)$$

Thus, we can justify the algorithm in (13) for computing the OPTIMUM action after observations o_1 through o_n according to a naive Bayes model *m*.

```
(13) a   OPTIMUM $m, o_1 \dots o_n$
             for each $j$: $p'_j \leftarrow P_m(S=s_j) * \prod_i P_m(O_i=o_i|S=s_j)$
             $p_o \leftarrow \sum_j p'_j$
             for each $k$: $u_k \leftarrow \sum_j (U_m(a_k, s_j) * p'_j / p_o)$
             $d \leftarrow$ any $k$ that maximizes $u_k$
             RETURN $(a_d, u_d)$
```

By (11) and (12), the algorithm ensures that the value of p'_j/p_o is the posterior

$$(14) \qquad P_m(S=s_j|O_1=o_1\dots O_n=o_n)$$

Thus by (10), the algorithm stores in u_k the expected utility in *m* of action a_k.

(12) shows the sense in which the naive Bayes model's independence assumptions allow an agent to generalize from its experience by rule. An agent using (12) (as implemented in (13)) is making a prediction about a combination of observations o_1 through o_n that it may never have seen together before. The prediction

rests on the agent's interpretation of its experiences in terms of a small number of likelihood parameters, $P_m(O_i = o_i | S = s)$, each of which summarizes commonalities among the disparate experiences the agent has had. The rule that underlies the generalization is in this case quite simple: multiplication.

More formally, we now see how we can eliminate much of the complexity of a decision model in a naive Bayes model. To illustrate, we will use $|S|$ to represent the number of states in the model and $|A|$ to represent the number of actions available to the agent; we will again assume that the observations are binary.

The naive Bayes model determines a utility for each combination of observations and actions. This is $|A|2^n$ values in this case. In a decision model, this would be prohibitive; each value would have to be represented, and learned, separately. But in the naive Bayes model, these $|A|2^n$ values are derived indirectly, using a number of parameters that is potentially much smaller. The naive Bayes model has a table of utilities with $|A||S|$ entries; it has a table of priors with $|S|-1$ entries; and it has n tabled likelihood functions, each with $|S|$ entries. That is a total of $(|A|+n+1)|S|-1$ parameters.

We may be able to identify a reasonable number of meaningful states in the world as part of designing our agent, so that $|S|$ is very small by comparison to 2^n. Then the naive Bayes model offers a vast improvement in size; there is a corresponding improvement in the amount of memory required to represent the model and the amount of training data required to build it.

Of course, not all environments can be described in terms of a single variable that takes on a small number of values. Richer environments demand richer models which offer an agent ways to reason about the state of the world more systematically. Indeed, effective modeling may mean understanding at a high level how and why an environment works, and realizing that explanation in computational terms; an agent may need this high-level explanation to select the actions that it should take based on the observations, and more generally on the experience, available to it.

Fortunately, this modeling need not start from scratch. Designers have ready sources of causal hypotheses, from their scientific understanding of the real world, from evaluations of prior agents, and even from intuitions about our own action in the world. At the same time, AI research offers an inventory of broadly useful models that implement common forms of explanation. Important special cases come from domains where agent need to recognize patterns of change over time or patterns of hierarchical structure in sequences of observations. Such models can be built and used effectively with elegant and efficient algorithms. But graphical representations of probabilistic dependencies, as illustrated in Figure 10, can describe arbitrary causal interactions in an environment.[24] Not surprisingly though, as models become more general, it becomes increasingly difficult to tie them to the training available in concrete domains or to algorithms that enable tractable decision-making. Research proceeds.

5 Directions, Connections, Distinctions

The enterprise of modeling the world is open-ended, and so is the enterprise of AI itself. In a chapter such as this, I cannot even begin to catalogue all possible problems, all possible models, even all possible approaches. Rather my aims have been to introduce the goals and perspectives of AI practice, in Section 2; to acquaint you in detail with a concrete case of AI practice, through Section 3; to hint at the problems and possibilities to which past AI research has led, in Section 4; and, perhaps with all this, to point out the general problems of real-world action that AI shares with cognitive science, and so to whet your curiosity for the additional results—indeed the future results—which AI research still offers.

These additional results include the numerous approaches to intelligent behavior based purely on optimization to which I have given short shrift. These engineering approaches are effective and useful, and historically have been just as inspired by biological computation—and just as inspiring in the effort to understand biological computation—as the more severe model-based techniques I have emphasized.

I have also passed over a rich tradition of purely logical models of the world. The motivations for such models have always been diverse (and contentious); for me, such models contribute a worst-case analysis of intelligent behavior. Logic demands that we avoid risk altogether by formulating models and policies that must succeed, within limits, no matter how the world might turn out to be. This worst-case design provides the only principled way to back off from dependence on distributions derived from empirical data to strategies that apply in genuinely novel situations—that is, even in the most remote possibility.

The diverse investigations of AI aim for agents that are sensitive to empirical regularities in their surroundings and that exploit these regularities efficiently to thrive in their environment. These are strengths that biological intelligence also exhibits. But AI research can base its strategies and computations on designs that depart radically from biology; as it does so, it increasingly delivers intelligent real-world behavior in ways that diverge from biological intelligence. For example, in its chess match with Kasparov,[25] IBM's Deep Blue relied on hardware and software that was capable of performing billions of operations per second and that in fact pursued extravagant search forward move-by-move through possible developments of the game; such a strategy seems impossible for a brain that makes up in parallelism what it lacks in speed. Mismatches proliferate among the many important problems which people master only with difficulty: medical diagnosis, investment, scientific data analysis, engineering design.

Such examples underscore the uniqueness of AI as an engineering approach to building computational artifacts that act in the real world. Practitioners of AI must take this engineering perspective to heart to have an impact on the way people design and build useful agents. Cognitive scientists must also come to grips with it if they are to exploit new ideas from AI research which promise to inform our scientific study of ourselves.

But all should welcome the rigor that the view inspires. In engineering, an *ad hoc* implementation—however ingenious—is quickly discarded as problems and technology change; a principled design—a model of the world and the computational results to put the model to use—remains a lasting contribution. In cognitive science, theories of the functional value of human cognitive abilities will remain speculation until connected with an engineering model of the problems people face and the performance that people achieve in everyday circumstances. Inspired by such inherent connections, I for one expect the long and fruitful interchange between AI and cognitive science to continue.

Thanks
Doug DeCarlo, Karin Johnsgard, David Parkes, Lou Steinberg.

Notes

[1] With his paper *On computable numbers, with an application to the Entscheidungsproblem* (Proceedings of the London Mathematical Society, 1937), Alan Turing offered his celebrated characterization of all mechanical procedures which can be executed according to a finite set of precise instructions.

Noam Chomsky's *Syntactic Structures* (published 1957) and the more incendiary *A review of B. F. Skinner's* Verbal Behavior (Language, 1959) decisively introduced linguistics, philosophy and psychology to a computational interpretation of the rules of linguistic structure.

Alan Newell and Herbert Simon's *general problem solver* (implemented 1957 with programmer J. C. Shaw, and described in a contribution to Feigenbaum and

Feldbaum's *Computers and Thought*, 1963) invoked an explicitly symbolic conception of computation as a general way to derive goal-directed real-world behavior by algorithmic processes.

[2] David Hilbert stormed through mathematics at the end of the nineteenth century and beginning of the twentieth; he published groundbreaking work in algebra, number theory, geometry, and the foundations of physics and mathematics. But he concluded his career with investigations into proof theory whose goal was the demonstration of the consistency and completeness of mathematical method; the theory of computation, as developed by Kurt Gödel, Alan Turing and others, dissolved this goal into impossibility.

Leonard Bloomfield's text *Language* from 1933 set the agenda for American linguistics for the next twenty-five years. The book is remarkable both for its meticulous methodological advice, which helped a generation of fieldworkers document the words and sounds of the indigenous languages of North America, and for its frank admission that, to study such matters as linguistic meaning, linguistics awaited some new scientific tools (in retrospect, that tool was computation).

B. F. Skinner's *The Behavior of Organisms* (published 1938) advanced a particular kind of associative learning, the pairing of stimulus and response through reinforcement, as a uniform explanation for the results of experiments documenting the ability of animals to act successfully in problematic environments. Skinner's behaviorist approach left no place for psychological representation, or for computation, in the explanation—an omission that ultimately proved the theory's undoing.

[3] A revised version of this chapter will appear in the second edition of Zenon Pylyshyn and Ernie Lepore's *What is Cognitive Science*, to be published by Basil Blackwell next year.

[4] The landmark implementation is described in W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The Interactive Museum Tour-Guide Robot," Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), 1998.

[5] Rudolf Kalman introduced this discrete-time filter in his paper "A new approach to linear filtering and prediction problems", in *Journal of Basic Engineering—Transactions of the ASME, Series D*, 83(1), pages 35–45, 1960. It was used right away in aerospace, for example for navigation in the Apollo space program. But its heavy burden of computational operations and numerical precision meant that the filtering process typically had to be approximated and simplified to run on available hardware; see for example Peter Maybeck's text *Stochastic Models, Estimation, and Control*, Academic Press, 1978.

[6] Newell's perspective, articulated for example in *Unified Theories of Cognition* (Harvard University Press, 1980), suggests an intimate relationship between AI and computational perspectives on human cognition. In particular, it postulates NATURAL representations—mental tokens that take on meaningful correspondences with the world but at the same time are manipulated syntactically through symbolic computation—to account for INTENTIONALITY, the connection between our own mental life and the broader world we think about and act in.

[7] With his theory of signs, Charles Sanders Peirce (1839–1914) offered an account of the parallels and diverges between linguistic meaning and natural meaning. Linguistic meaning is symbolic; the word *fire* for example is connected with fires in an arbitrary way. Smoke also means fire, but smoke is what Peirce called an INDEX: such signs are linked to what they mean by correlations in the natural world. See *The Essential Peirce, Volumes 1 and 2,* Indiana University Press, 1992 and 1998.

[8] *Making Hard Decisions* (Robert Clemen, Duxbury Press, 1995) is a representative text on decision analysis—for the MBA curriculum.

[9] See for example Ronald Arkin's *Behavior-Based Robotics*, MIT Press, 1998.

[10] A fraction of 0.8 of histories with this policy lead to an observation of $X = y$, to following, and then to a utility of 0.7; a fraction of 0.2 lead to an observation of $X = n$, to returning home, and then to a utility of 0.25. Weighting these outcomes by their probability gives $0.8 \times 0.7 + 0.2 \times 0.25 = 0.61$.

[11] At choice 1, the other options' utilities of 0.2 and 0.3 are less than the 0.61 we get by waiting and continuing with (1). At choice 2, our policy of following for 0.7 beats the alternative of returning for 0.25. At choice 3, our policy of returning for 0.25 beats the alternative of following for 0.1.

[12] Indeed, after studying this section, you might profit from constructing your own definition of its two key representations, the MODEL and the POLICY, and from accomplishing your own implementation of its two key algorithms, OPTIMUM and EXECUTE—either in a familiar language or even in one of the specialized languages of AI, such as Prolog, Scheme or SML.

[13] Here is one such definition:

(15) a In a model $rwd(u)$, policy *end* has utility u.

b In a model $obsv(X, [(v_1, p_1, t_1) \ldots (v_n, p_1, t_n)])$, policy $switch(X, [(v_1, \pi_1) \ldots (v_n, \pi_n)])$ has utility

$$\sum_i p_i * u_i$$

where u_i is the utility in model t_i of policy π_i.

c In a model $dcsn(n, [(a_1, t_1) \ldots (a_n, t_n)])$, policy $do(a_i, \pi)$ has utility u_i where u_i is the utility in model t_i of policy π.

[14] Just refer to (15).

[15] Here's one specification of EXECUTE.

(16) a To EXECUTE *end*, nothing is required.

b To EXECUTE $switch(X, [(v_1, \pi_1) \ldots (v_n, \pi_n)])$

```
    $o \leftarrow$ GET $X$
    $j \leftarrow$ the $i$ for which $o = v_i$
    EXECUTE$\pi_j$
```

c To EXECUTE $do(a, \pi)$

```
    RUN $a$
    EXECUTE $\pi$
```

[16] At the first step of action, there is one outcome; after each step of observation, the number of outcomes doubles; by the time the policy terminates, the leaves number $2 * \ldots * 2$ (n 2s); thus, 2^n.

[17] This fact admits two kinds of explanations. Conceptually, we can see each layer i of observation in the model as specifying a distribution of variable X_i that is conditional on specific values of any earlier observations X_1 through X_{i-1}; this distribution is written $P(X_i = x_i | X_1 = x_1 \ldots X_{i-1} = x_{i-1})$. In general, any joint distribution on multiple variables (perhaps conditioned on some information C) can be factored as the distribution of the first (given C) times the distribution of the others given the first (and C), as in (17):

(17) $$P(X_1 = x_1 \ldots X_i = x_i | C) = P(X_1 = x_1 | C) P(X_2 = x_2 \ldots X_i = x_i | X_1 = x_1, C)$$

Unfolding (17) $n - 1$ times, then, we see that the distribution provided at each layer is exactly what is needed. More algebraically, and more significantly for present purposes, the joint distribution is specified by $2^n - 1$ parameters and the model t also has exactly $2^n - 1$ free parameters for probabilities (the counts reflect the fact that probabilities always sum to 1). Matching these parameters is thus a matter of solving appropriate equations.

[18] The standard introduction to machine learning is Tom Mitchell's *Machine Learning*, McGraw-Hill, 1997.

[19] See Nello Cristianini and John Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, 2000.

[20] See Christopher Bishop, *Neural Networks for Pattern Recognition,* Oxford University Press, 1995.

[21] See Richard Sutton and Andrew Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[22] The famous formulation—in which Chomsky argues that people's productive use of language must be informed not only by experience with language but also by models of language embodying substantive universals—is in the collection *Language and Learning: The Debate Between Piaget and Chomsky* edited by Massimo Piatelli-Palmerini and published by Harvard University Press, 1980.

[23] The result, first explored by minister and sometime mathematician Thomas Bayes (1702–1761), is easy to rederive. Just start from the two ways to express the joint probability of two events, a possible cause c and its possible effect e:

$$P(c,e) = P(c|e)P(e) = P(e|c)P(c) \tag{18}$$

Divide by $P(e)$, to see how to infer the cause c given a model of its effects:

$$P(c|e) = \frac{P(e|c)P(c)}{P(e)} \tag{19}$$

[24] The seminal monograph is Judea Pearl's *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.

[25] Documented on the internet at http://www.research.ibm.com/deepblue/home/html/b.html.