Anticipatory Artificial Intelligence

Michael Rovatsos

School of Informatics, The University of Edinburgh
Edinburgh EH8 9AB, United Kingdom
mrovatso@inf.ed.ac.uk

1 Introduction

Anticipation occupies a special role in Artificial Intelligence (AI). As AI strives to replicate human intelligence in artefacts that utilise digital computing machinery, it involves multiple, interdependent processes of predictive modelling, some of which occur on the side of the designer at *design time* (when the system is built), while others take place within the artefact itself at *runtime* (when the system operates in its environment).

From the designer's perspective, in designing an AI system, one is typically interested in building a computational system that can solve a class of problems in a given domain and whose solution is assumed to require intelligence. The aim is to achieve a performance similar (or superior) to that of humans along some pre-defined metric and/or to perform the task in ways that resemble the ways in which naturally intelligent systems would proceed. Even without making any assumptions regarding the anticipatory nature of this design process, it is inevitable that the designer have a model of the *system* tasked with solving a class of problems (as this will have to be eventually encoded on some machine), a model of the *problem domain* (since problem instances will be encoded as input to this system), and a model of the *cognitive process* humans/animals apply to the same class of problems (as this is what the system is supposed to replicate). But given that there is a *teleological*, future-directed rationale for designing AI systems (either solving a real-world application problem, or learning more about the nature of intelligence), we have to consider the anticipatory nature of this process. Then, the issue becomes one of choosing *how* to design the AI system based on the anticipated consequences when viewing the system as an *effector* that impacts the future state of the designer's world [43], not only in achieving the designer's primary objectives, but also considering any unintended side-effects, including any considerations regarding the safety, ethical behaviour, and social impact of the system.

Looking at the AI system itself, while theoretically – if we were to replicate all of human intelligence in machines – anticipation would be indispensable, it is certainly not necessary to build anticipatory processes into every such system. In fact, champions of purely *reactive* systems [7, 8] have claimed that no explicit, predictive models of the world are needed for AI systems to achieve intelligence, and some very recent advances in areas such as reinforcement learning and deep neural networks focus on *strong* (i.e. implicit) anticipation, e.g. to recognise relevant actions using attention [35] or to anticipate the occurrence of future visual representations in video feeds [59]. There are even researchers who view the explicit, so-called *weak* model-based anticipation that is of most interest in the interdisciplinary study of anticipation [38] as potentially less useful for computational models of cognition, given that any realistic model of the problem environment might be way too complex to explore exhaustively to base decisions on predictions [42]. Nonetheless, if we *do* want to focus on anticipatory AI, we need to look at the main techniques that involve predicting events, planning complex courses of

actions, assessing alternatives in the face of uncertainty, and revising one's decisions flexibly to unforeseen circumstances.

This chapter aims to provide an introduction to anticipatory AI techniques, while linking the anticipatory capabilities of the *systems* developed using AI techniques to the capacity these methods afford the *designers* of these systems to anticipate their behaviour. This approach is motivated by the belief that an understanding of these two types of anticipatory processes has the potential to enable us to establish a more comprehensive methodological basis for studying anticipation in AI.

As a motivating example, consider the recent success of the AlphaGo system [51] in achieving human-level performance at the game of Go, a game traditionally considered notoriously difficult for computer programs. An important aspect of the design of this system is that it was first trained on 30 million positions from games played by human experts to predict human moves, and then improved its strategy by playing millions of games against itself, simulating human behaviour. The internal model of the game the system uses allows it to assess the quality of a move at any given point in time based on an anticipatory process that evaluates the estimated quality of future game positions likely to result from this move. In other words, it anticipates the future consequences of its possible moves by predicting the behaviour of its opponent based on patterns derived from observation of human experts and a simulation of their behaviour in situations not observed in historical game data.

The anticipatory process applied by the designers of the system operates on a different level: It is based on the assumption that a large dataset of games played by humans is representative of the behaviour of expert players, and that the amount of additional simulated games the system played against itself is sufficient to explore most game situations and opponent strategies that might realistically occur. AlphaGo's designers also assumed that testing the system against a certain number of fictitious opponents would be sufficient to allow them to anticipate its performance, and thus to give them a good idea of how well it might perform in a real tournament against a human champion. This involved a conscious decision to establish when the system was "good enough" and its training was complete. Of course there were also more basic assumptions embedded in the system's design: The game of has only certain valid moves, possible board configurations that can result from these moves, and hence alternatives that have to be considered. The outcome of each move is deterministic as Go involves no element of chance, and all of the system's decisions can be executed correctly and completely when playing the game.

At both levels, we observe the typical elements of an anticipatory process: a model of the system is used to consider different alternatives about what might occur in the future, and make decisions about what action to take in the present. And, the future is seen as a projection of the past through the present [39].

What is maybe unique to anticipation in AI as opposed to other disciplines is the importance of *assumptions* and how they are embedded in the design of the system in a way that links the system's anticipatory properties to those of its designer. Since a lot of the predictive "work" will be performed by the system itself *in lieu* of a human anticipatory process, the issue of how to frame and bound the different futures the system should consider through appropriate assumptions becomes central. As we will see below, such assumptions often take the place of actual prediction of the system's behaviour. Rather than actively engaging in predicting the behaviour of the AI system, we expect that if its model of the world is correct, its own reasoning will be a faithful representation of the anticipatory processes a human would employ.

We hypothesise that is a consequence of the very nature of anticipatory AI and the belief that if an AI system correctly implements human-like anticipatory behaviour, this would make explicit prediction of (and critical reflection on) possible AI behaviours unnecessary. If true, this would have interesting, but potentially also dangerous methodological consequences. Below, we will survey basic AI techniques, explaining for each case, the correspondence between the anticipatory methods applied by human designers and those embedded in the AI systems that result from use of this method.

The remainder of this chapter is structured as follows. In section 2, we provide some basic background on computational modelling, mainly aimed at readers without a background in computing methods. Section 3 introduces a number of core AI techniques, commenting on the role of anticipation within them. In Section 4, we turn our attention from individual AI algorithms toward intelligent and autonomous agents, discussing what additional elements of anticipation are embedded in notions of autonomy and in the specification of objectives for these agents. Section 5 considers the issue of anticipating the behaviour of AI systems and the broader impact of AI. Section 6 concludes.

2 Computational Modelling

Anticipation relies on models to capture the nature of a system, both the computational artefact and the environment or problem domain within which it will operate. To characterise the space of available models, we need to specify the language used to describe what states a system can be in at any particular system, and how the system changes state as a function of present and past states and the forces imposed on it from the outside [30].

Methodologically, AI borrows fundamental models of information processing from computer science, adopting the "standard model" of computation on digital machinery. There are three fundamental viewpoints of analysis that can be applied to this model: from the point of view of *data*, the main question is how input is transformed to output using a computational procedure; from the standpoint of *algorithms*, the question is what set of instructions this computation procedure may consist of; and taking the standpoint of a *machine*, emphasis is put on what types of artefacts may process these instructions to perform this computation.

These three viewpoints provide the fundamental buildings blocks of modelling in computing and AI, in the sense that, to be considered "computational", any model of a system has to specify what data is made available to it and how this is processed, using operations that can be performed on standard computing machinery. The machine model representing digital computers is that of the Turing Machine [58], an abstract model of a machine that can read and write symbols on an infinite tape based on a finite set of rules. In reality, no computer has infinite storage, but computers are programmed "as if" this were the case, which creates no major problems as long as we do not require more storage than is actually available.

For the purpose of this chapter, however, we will assume a much simpler model, that of a Finite-State Machine (FSM) [25], which is sufficient to capture any system that can only be in a finite number of different states. A very simple example of an FSM might be a hotel safe that can be locked with a 4-digit code. The safe has three states, it can be locked, unlocked, or ringing the alarm. Entering the correct code will lock/unlock it, entering the incorrect code will make the alarm sound. We can describe this FSM in diagrammatic form, using a *graph* describing a *state transition system*, where edges (arrows) are labelled by inputs (the data processed by the system) and connect nodes (circles) representing states, such that the edges describe the transitions between states. The diagram for our FSM is shown in

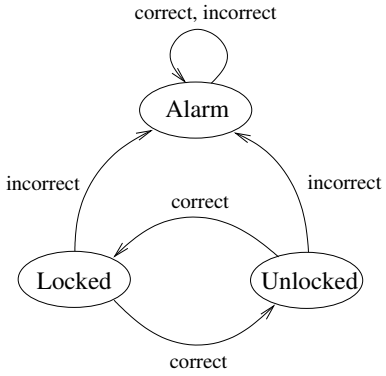

Fig. 1. A simplified finite-state diagram describing our safe example

figure 1. There exist different notational variations for these diagrams, where outputs may either be associated with the edges or with states, but these are mathematically equivalent. In this example, we will assume that entering a new state is also associated with producing an output, i.e. sounding the alarm.[1]

The first thing to observe about such models is that they involve determining the range of states considered, which depends on what aspects of the system, including its environment, are taken into account. Often, these are expressed through system variables that can take on different values, and the possible states of the system is determined by considering all possible combinations of values those variables can assume. For example, we might introduce a variable `state` that can either be `locked`, `unlocked`, or `alarm`, to denote that the alarm will sound. The `input` to the system (viewed as another variable describing the current state of the world) can have two values, `correct` or `incorrect`. With this, we obtain $3 \times 2 = 6$ combinations of what situation the system might be in at any time. Considering all these combinations is essential to determine the so-called *combinatorial complexity* of a computational problem, as it determines how hard it may be to compute the desired outcome. The larger the number of possible states, the longer it may take to achieve a desired state.

For any but the most simple systems, specifying the behaviour of a system by drawing such diagrams is not efficient, and programming languages are used instead, which specify how system variables are manipulated through a number of instructions, taken from a fixed set of instructions specified by the *syntax* of the language. The following is an example of what a program for the safe described above might look like:

```
while(true) {
  if(input==correct) then
     if(state=locked) then
         state=unlocked
     if(state==unlocked) then
         state=locked
  if(input==incorrect) then
         state=alarm
}
```

[1] Formal definitions of FSMs normally also involve an initial state that the system starts off with, and a set of final states where its operation would terminate, but we can ignore these for our purposes.

This contains the key elements of many programming languages, such as assignments (e.g. `state=locked`) which change the value of a variable in the system's memory, conditions (e.g. `input==correct`) that involve inspecting a variable's value, conditional statements ("`if` a condition is satisfied `then` perform some action") and loops (e.g. `while(true)`) that instruct the system to repeat something under certain conditions. In our example, `true` is a constant that is always true, so the part between the curly brackets will be repeated forever.

What our very simple program basically states is *"repeat forever: if the correct code is entered, unlock the safe if locked and vice versa, and if the incorrect code is entered, sound the alarm"* (note that this implies that entering the correct code will never return the safe into locked/unlocked (non-alarm) state, unlike in the FSM of figure 1).

It is important to emphasise that any such computational model involves a conscious abstraction of reality, as this has important consequences for anticipation, both for the designer and for the system. This process of abstraction manifests itself in different ways. Firstly, the state space introduced makes specific distinctions between different situations that could be made at various levels of granularity. In our example, we do not consider, e.g., how a user might enter the digits individually, or keep track of intermediate states in the process until the code is detected to be correct or incorrect. Secondly, by including certain variables we invariably exclude others. For example, we choose to ignore whether there is any money in the safe. If the safe had a sensor to check whether it is empty, sounding the alarm might be considered unnecessary whenever it is, in fact, empty. Thirdly, all transitions, inputs, and outputs are deterministic, instantaneous, and error-free – there is no modelling of any uncertainty that input signals can be detected, that their content can be determined unequivocally, that time passes between different steps in the computation, or that transitions may fail.

Modern computer science provides methods to express all of these additional aspects, e.g. by labelling different transitions with probabilities denoting the likelihood that certain events will occur or observed variable values will be correct, introducing a notion of time, or modelling "hidden" system variables that cannot be directly observed. What is important, however, is that any refinement of a model invariably increases its complexity, both in terms of the size of the *representation* of the problem environment, and in terms of the *complexity* of solving a specific problem, arising from the number of combinations of states and actions implied by a certain model.

At this point, we should explain more precisely what we mean by "solving a problem" in this context. Generally speaking, a designer will aim to define a system that achieves a certain behaviour, i.e. computes certain outputs on certain inputs. In the example above, we want to allow the user to lock and unlock the safe if they enter the correct access code, and sound the alarm otherwise. The FSM and program we have given above define representations for the input and output corresponding to real-world entities[2] and constitute algorithms that specify how outputs are generated from inputs. While our example only solves a very simple and specific problem, there are algorithms that can operate on a whole range of inputs (e.g. for sorting a list of numbers, finding a route on an arbitrary geographical map, etc).

In AI, as we are interested in tasks that involve elements of human intelligence, we generally expect a degree of flexibility and complexity that is much higher than that of a simple

[2] In reality, we also have to consider that these somehow have to enter the system through sensors and actuators by way of physical coupling of its computational components to the locking mechanism, the keypad, and the alarm bell, but we omit these details from our discussion.

algorithm. In the following section, we review the principal categories of AI techniques, focusing on the ways in which they embed anticipation, both at design- and at run-time.

3 Anticipation in common AI techniques

This section reviews some of the most widely used techniques in AI, discussing what anticipatory elements they embed. This survey is not intended to be exhaustive or to introduce specific algorithms in detail, and it does not make explicit reference to major areas that focus on specific types of systems, e.g. robotics, machine vision, or natural language processing[3]. Instead, it considers the main categories of methods deployed in the design of AI systems, distinguishing between the problem formulations they involve, and focusing on the relationship between the designer's anticipation capabilities, how they affect those of the system's, and what we can say about what behaviour human designers can anticipate of the system when deployed in the real world.

3.1 Search-based problem solving

One of the most general approaches to solving complex problems using generic algorithms is *search* [13]. The underlying problem formulation is based on considering the different possible solutions to a problem, and systematically considering each of them one by one in order to find one that constitutes a solution.

A search problem is defined by a *graph* that is explored step by step by generating *successor states* from previous states explored, such as that shown in figure 2.

For example, when attempting to find a route from location x to location y in a map, the search might start from node x and generate all its immediate neighbour locations, then check whether any of these neighbours is y (this is called the *goal test*), and continue exploration starting from one of x's neighbours in the next step. The choice of which state/node to continue from in the next step is determined by the *search strategy*, and using the right strategy may significantly affect how many steps are needed (the so-called *time complexity* of the algorithm) or how many nodes need to be stored during the search (the so-called *space complexity* of the algorithm). *Heuristics*, i.e. rules of thumb that are supposed to speed up the search process, are often used in so-called *heuristic search*, though the "shortcuts" they suggest may make the algorithm incomplete (i.e. not able to identify a solution even though one exists) or sub-optimal (i.e. perform more computation steps than are actually needed).

If there is only a finite number of states, and the algorithm avoids re-visiting states it has already considered or using incomplete heuristics, standard search algorithms exist that will always find the solution, simply by making sure that every state is visited eventually.

Search algorithms, like many other AI methods below, aim to be formulated in a *domain-independent* way, i.e. we want to be able to specify the algorithmic procedure once, and then apply it to many problems, given an appropriate encoding of the problem. In the route planning example, we could identify the state by the name of the current location, `current_location` == `Edinburgh`, and use a database that contains a list of pairs of cities neighbouring each other, e.g. `neighbours(Edinburgh)` = {`Aberdeen`, `Glasgow`, `Newcastle`} to add successor states incrementally in the search graph.

Using a broad range of such possible encodings, standard search algorithms can be applied to any problem that can be formulated in terms of a discrete (i.e. containing an enumeration

[3] The reader is referred to general textbooks on AI, for example [45], for a more comprehensive treatment.

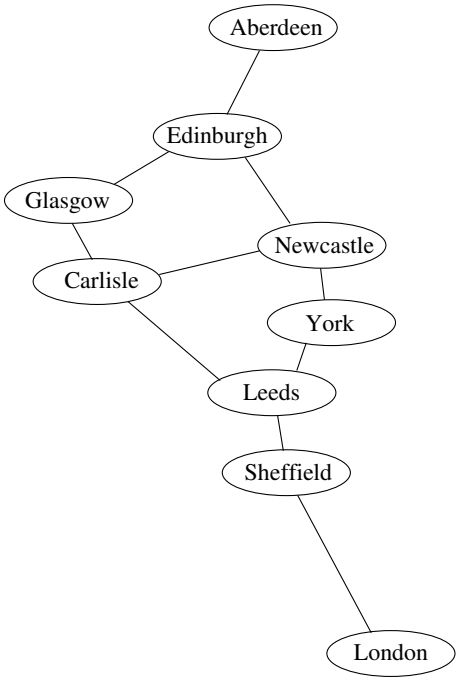

Fig. 2. A search graph representing a route map that connects major cities of the United Kingdom. Edges are undirected, as the can be travelled in both directions. The topology of places is depicted to roughy correspond to their actual geographical positions, though a search algorithm may not have this information.

of distinct elements that can be iteratively constructed) state representation and a successor-state operation that will generate states immediately adjacent to any given current state. For example, we could use search to explore all combinations for the code of the safe in our example from the previous section, all possible states that could occur in a game of chess, or all expressions that could result from mutation and reproduction in a gene. In many cases, we will be interested in the *path* to a solution state (e.g. a route from A to B in the route planning example, a winning strategy in chess), which then provides a recipe for action the system could implement to solve the problem.

This domain-independence is generally considered to be an aspect of intelligence of the system, in the sense that once implemented, the same program could deal with a broad range of concrete problems, in much the same way as humans may apply similar exploration of all alternatives when performing a task. The intelligence may also lie in the heuristics used. For example, when trying to plan a trip from A to B generally we will be trying to look for intermediate waypoints that get us closer to the destination in terms of straight-line distance, rather than considering all possible directions in every step of the way.

Search is based on the assumption that the transitions from one state to another are fully captured by the problem formulation (e.g. all neighbouring locations are listed in our database), that the effects of a transition are fully known and can be reliably anticipated (e.g. in the formulation above we never run out of fuel, and we never end up in an unexpected location), and that there is, in principle, unlimited computational resource to find a solution regardless of how big the space of possible states might be. The anticipatory capabilities of the system are bounded by these assumptions, in that they delimit the set of possible future

states in their consequences, and ignore any potential factors that have not been accounted for, including any side-effects of deciding to actually enact what the solution dictates.

Some of these assumptions can be relaxed by using more advanced problem formulations, for example by introducing non-determinism (e.g. by placing probabilities against the edges of the search graph to express, for example, that the likelihood of the car breaking down at any point while driving is 5%, and considering all possible strategies so as to pick the safest one) or by limiting the number of states explored before making a decision (e.g. looking only four moves ahead in a chess game, and then deciding what to play based on an evaluation of the states reached even if they do not clearly indicate we have won or lost the game).

Nonetheless, the *domain theory* (the model of the world any given problem formulation reflects) is assumed to be correct, meaning that there is no way of *modifying* it at runtime. This is because, in some ways, search-based problem solving is *purely* anticipatory within the limits of the model it employs – it provides no facility for *reacting* to any circumstances that may arise while attempting to enact its decision at execution time in the real world (e.g. actually drive from A to B).[4] In section 4 we will discuss what role *reactive* properties of AI systems play in relation to their *proactive* (future-directed, anticipatory) properties when judging the intelligence of these systems.

When looking at the ways in which the designer can anticipate the system's behaviour at run-time, there is an important distinction we have to make, and which also concerns the other methods described below: If all aspects of the system are known, and the system's operation is not affected by events occurring at execution time, then the behaviour of this system can be entirely anticipated *in principle*. However, due to the complexity of systems that solve hard problems, this will not be possible *in practice* in most real-world cases, as it is simply not cognitively possible for a human to predict all possible search steps in a large state space. In fact, in many applications for which AI systems are used, the very reason for creating the system is to manage a complex solution space that a human cannot reason over. We will return to a discussion of this interplay below in section 4.

3.2 Planning

Above, we have given examples of how search can be used to determine courses of action to achieve a certain goal from a starting state. This is the focus of the area of automated *planning* [22], which often utilises search as an algorithmic process to find a path from one state to another, but involves more elaborate representations of states and actions that allow for a far greater degree of flexibility.

Representations used in planning are used to encode *action theories*. Such action theories describe the circumstances under which an action can be performed, and how it modifies the state of the world when performed. In the classic STRIPS formulation of planning [15], for example, an action is described by *preconditions* and *effects*, where effects are described using an *add-list* (to add certain facts about the world after the action is executed) and a *delete-list* (to remove facts no longer true after the action is performed).

Let us extend our route planning example from the previous section to a scenario that describes how a car, when driven by a human driver, moves between locations. This scenario may involve a number of action *schemata*, i.e. abstract specifications of the preconditions and effects of different types of actions, e.g. driving from one location to another, re-fuelling the

[4] Note that when referring to execution time, this should not be confused with runtime, i.e. when the algorithm (rather than the solution it returns) is run to determine the solution.

car, etc. A schema $\texttt{drive}(\texttt{c}, \texttt{d}, \texttt{a}, \texttt{b})$ might be used to express that driver d moves car c from a to b. Sensible preconditions for this action might be

$$\texttt{driver}(\texttt{d}), \texttt{location}(\texttt{a}), \texttt{location}(\texttt{b}), \texttt{at}(\texttt{c}, \texttt{a}), \texttt{at}(\texttt{d}, \texttt{a})$$

and the effects could be specified as

$$-\texttt{at}(\texttt{c}, \texttt{a}), -\texttt{at}(\texttt{d}, \texttt{a}), \texttt{at}(\texttt{c}, \texttt{b}), \texttt{at}(\texttt{d}, \texttt{b})$$

where the operator "$-$" indicates that this fact has to be deleted after the action is performed, and facts that do not contain this symbol should be added to the current state. This specification would be intended to express that for the action to be performed, `d` needs to be a driver, `a` and `b` locations (you cannot drive between objects that are cars, for example), and both `c` and `d` are initially located at `a`. After the action is performed, they would no longer be at location `a`, and instead both be located at `b`. None of the other facts are affected by the action, for example, `d` is still a driver, and `c` is still a car, after the action is complete.

At this point, it is worth providing a bit more detail regarding the notation we have used in the example. First of all, facts of the form $\texttt{p}(\texttt{a}_1, \ldots, \texttt{a}_k)$ are written as *predicates* (with a predicate name `p` and arguments $\texttt{a}_1$ to $\texttt{a}_k$, representing objects or variables), which borrows from conventions used in mathematical and computational logic (see subsection 3.3 below), but essentially is simply used as a query against the variables of the current state. Recalling our treatment of state variables in section 2, the STRIPS notation assumes that the current state is expressed using a *knowledge base*, which is essentially a database of properties of the current state. In our example, it might contain the following facts:

$$\{\texttt{driver}(\texttt{Alice}), \texttt{car}(\texttt{Herbie}), \texttt{location}(\texttt{London}), \texttt{location}(\texttt{Edinburgh}),$$
$$\texttt{location}(\texttt{Glasgow}), \texttt{at}(\texttt{Alice}, \texttt{Edinburgh}), \texttt{at}(\texttt{Herbie}, \texttt{Edinburgh})\}$$

to express that driver Alice and car Herbie are in Edinburgh, and Edinburgh, London, and Glasgow are all locations in the system. When we pose a query against this knowledge base with a predicate that contains variables, e.g. $\texttt{at}(\texttt{x}, \texttt{y})$, the knowledge base would return all matching facts ($\texttt{at}(\texttt{Alice}, \texttt{Edinburgh})$ and $\texttt{at}(\texttt{Herbie}, \texttt{Edinburgh})$ in this case).

If we execute the action $\texttt{drive}(\texttt{Alice}, \texttt{Herbie}, \texttt{Edinburgh}, \texttt{Glasgow})$ (a concrete *instantiation*, or *grounding* of the general action schema that contains no variables), we will end up in a new world state

$$\{\texttt{driver}(\texttt{Alice}), \texttt{car}(\texttt{Herbie}), \texttt{location}(\texttt{London}), \texttt{location}(\texttt{Edinburgh}),$$
$$\texttt{location}(\texttt{Glasgow}), \texttt{at}(\texttt{Alice}, \texttt{Glasgow}), \texttt{at}(\texttt{Herbie}, \texttt{Glasgow})\}$$

as prescribed by the effects of the schema, which remove $\texttt{at}(\texttt{Alice}, \texttt{Edinburgh})$ and $\texttt{at}(\texttt{Herbie}, \texttt{Edinburgh})$, and add $\texttt{at}(\texttt{Alice}, \texttt{Glasgow})$ and $\texttt{at}(\texttt{Herbie}, \texttt{Glasgow})$. The preconditions of the schema ensure that actions such as $\texttt{drive}(\texttt{Herbie}, \texttt{Herbie}, \texttt{Edinburgh}, \texttt{Glasgow})$ or $\texttt{drive}(\texttt{Alice}, \texttt{Herbie}, \texttt{Glasgow}, \texttt{Edinburgh})$ are not applicable in the current state.

Planning algorithms enable us to determine what sequence of actions is necessary to reach a given goal (e.g. $\texttt{at}(\texttt{Alice}, \texttt{London})$) from an initial state like the one above, usually by performing some form of search exploring every action available in each step and terminating when any concrete state is found that satisfies the goal description.[5] Many variations of

[5] If Alice gets to London, lots of other things might still hold, but we only require this specific fact in this case, which implies we do not care about what else might be true in the final situation achieved by the plan.

planning formalisms exist that extend the basic STRIPS model by additional features, e.g. allowing for uncertain effects or effects that only occur under certain additional conditions, require certain things about the world *not* to be true for the action to be executed, or involve returning a plan that is not a sequence of actions but a more complex procedure that may require *sensing* facts not known from the outset to select subsequent actions.

Despite resembling search-based problem solving to some extent, automated planning affords additional capabilities that are essential to intelligent decision making. Crucially, it breaks down the notion of state into its different properties, which introduces a notion of *relevance* for different alternatives considered at any point in time in several ways. Firstly, actions not applicable in a given intermediate state are not considered at all, which saves the algorithm from looking at options that are not meaningful. Secondly, as each action only modifies certain aspects of the state, we can track evolution of this state along a sequence of actions by modifying the set of facts true after each step, rather than enumerating a (potentially huge) hypothetical state space. Thirdly, the search algorithm will try only to achieve those aspects of the state we care about in the goal description, which means that *any* concrete state that satisfies them counts as a solution, and it can attempt to make all the goal conditions true one by one. Finally, the facts that need to be satisfied provide very useful information for search heuristics that speed up the search process massively, as they allow for prioritising which actions to explore in each step much more systematically. This has helped develop algorithms that used to scale only to plans of a few actions 20 years ago to systems that can solve problems that involve thousands of actions nowadays.

What does the use of planning techniques imply in terms of anticipatory elements of intelligence on behalf of the designer, and the planning system? In terms of the basic assumptions regarding how much is known about the problem environment, and how predictable and certain the outcomes of actions are, planning is indeed very similar to search-based problem solving, and all the remarks made at the end of section 3.1 apply also to planning. However, planning adds an important element of anticipation related to the *frame problem* [32], commonly regarded as one of the fundamental problems in AI. The frame problem is concerned with making sure that the representation of a problem domain and the inference procedures over it capture all aspects of the world correctly, and attempts to address the fundamental impossibility of achieving this. It can be broken down into further sub-problems, such as the *qualification problem*, concerned with the impossibility of listing all the preconditions required for a real-world action to have its intended effect, and the *ramification problem*, concerned with the impossibility of listing all possible side-effects or indirect consequences of an action.

The implications of the frame problem are immediately obvious in the representations used by planning. In our example, we have made an *ontological commitment* to only consider certain aspects of the world when describing our action theory. For example, we have not required that the car is in good working order, or that it is not struck by lightning en route, and we have not captured effects like the reduction of fuel in the tank after driving. In terms of anticipation, it is impossible to consider all possible variations in circumstances and all possible side-effects when defining a concrete action theory. At the same time, the assumptions made regarding the *correctness* of a planning algorithm (stating that it will only output plans that will certainly achieve the goal from the initial state) and its *completeness* (stating that it will find a solution if one exists) imply that the designer, at the time of specifying the action theory, believes that successful automated planning reflects that the plan returned will be executable, and that it will lead to the desired outcome.

This nicely illustrates that when delegating a planning task to an AI system the anticipatory abilities of the designer are mirrored by those of the system, which will be no less and no more provident than the human who imparted it with the theory of causal change in the world she believes appropriate. Without additional machinery to repair faulty theories, they can only ever reflect the understanding of their designers in terms of accuracy and realism.

3.3 Knowledge representation and reasoning

Our treatment of planning already used a very simple form of *knowledge representation* [4] as it involved describing the world in terms of a vocabulary of facts and objects (called an *ontology* in AI terminology), and a specific form or *reasoning* in the form of deriving the truth value of facts in a state from their status in previous states and the assumed causal consequences of an action. Methods for representing and reasoning about knowledge are central to AI research in a much broader sense, and focus on the idea of being able to establish the truth of assertions regarding aspects of the world not immediately observed in current input data. For example, if we know that a car is at a certain location, then we know that its driver is at the same location, even if we only have data about the GPS location of the car, and no image of the driver sitting in the car. This type of information that is produced by *inference* from other facts and rules about the problem domain goes beyond the notion of retrieving stored data. If all we had added to a traditional database was the fact $\texttt{at}(\texttt{Herbie}, \texttt{London})$, a query of the form "what are all the things that are in London?" would simply return `Herbie` as an answer. Instead, querying a knowledge base that also contains the rule "if a car is in location X, its driver is also at location X", would return `Herbie` and `Alice`, if Alice is the driver of this car.

Being able to capture and utilise generalised knowledge of this form is seen as a key capability of intelligent systems, as it emulates the process of applying known properties of the world to new data to deduce information about *hidden* (including future) aspects of the world, just like humans do in everyday commonsense reasoning. In terms of anticipatory reasoning, we can in fact view knowledge representation techniques as an attempt to replicate the prototypical modelling relation as described by Louie [30] in a mathematical/computational system, which posits a coupling between *inferential entailment* in a *formal* system and *causal entailment* in a *natural* system (that is to be modelled) via *encoding* and *decoding* relations, which, in the case of computational systems, involve the formal languages these can process.

Two of the most common tools for representing knowledge and performing inference procedures on it are *logic* and *probability theory*. Logic-based methods are based on expressing knowledge in a formal language that permits syntactic manipulation in order to derive new knowledge. The fundamental contribution of common logical formalisms to AI is that they come with rules for such syntactic manipulation that preserve the *semantic* truth of the assumptions expressed in the logic. In other words, if the logical sentences we have written down to express our assumptions hold true, then we can guarantee that the *consequences* derived from these assumptions through a process of logical proof will be correct.

To return to our example, in the commonly used *first-order logic*, it would be captured by the following facts:

$$\texttt{car}(\texttt{Herbie})$$
$$\texttt{driver}(\texttt{Alice}, \texttt{Herbie})$$
$$\texttt{at}(\texttt{Herbie}, \texttt{London})$$

$$\forall \mathtt{c}, \mathtt{d}, \mathtt{l} \,.\mathtt{car}(\mathtt{c}) \land \mathtt{at}(\mathtt{c}, \mathtt{l}) \land \mathtt{driver}(\mathtt{d}, \mathtt{c}) \Rightarrow \mathtt{at}(\mathtt{d}, \mathtt{l})$$
$$\forall \mathtt{x}, \mathtt{l}, \mathtt{l}' \,.(\mathtt{at}(\mathtt{x}, \mathtt{l}) \Rightarrow \neg \mathtt{at}(\mathtt{x}, \mathtt{l}')) \lor \mathtt{l} = \mathtt{l}'$$

The first three lines here express our assumptions that Herbie is a car, Alice drives Herbie, and Herbie is in London. The fourth statement is a more complex rule that says "for any entities c, d, and l, it holds that if c is a car, c is located at location l, and d is the driver of c, then d is also located at l" (for the purposes of this example, we only call someone a driver if they are currently driving the car). The final rule states that "for all x and any two locations l and l', either x is located at l and not at l', or l and l' are the same location". Chaining the two final rules would allow us, for example, to infer that Alice is not in Edinburgh if she is in London, and demonstrates that more complex lines of reasoning can be performed if we use algorithms that can reason over such knowledge bases.

Computational logic [19] is a broad field that is also relevant for many areas beyond AI, but as far as using it practically to embed anticipatory methods in AI systems, the expressiveness of first-order logic (roughly speaking) represents the limit in terms of what is *computable*, i.e. there is an algorithmic procedure that will terminate when trying to answer any arbitrary query over a finite knowledge base. In fact, even in the case of propositional logic, a simpler variant of first-order logic that does not allow variables and quantifiers like "for all" or "there exists" along with some other features, there are significant challenges regarding the computational complexity of such proof procedures. In the worst case, a propositional logic query may require an amount of time that is exponential in the number of symbols used in a knowledge base to return a result. It is for this very reason that more constrained methods like planning, which borrow certain ideas from logic but restrict the types of knowledge that can be expressed and the queries that can be answered, are much more commonly used in real-world implementations of AI techniques.

Probabilistic reasoning [37], which makes use of mathematical probability theory and methods that originate from mathematical statistics, has grown immensely in popularity over the past thirty years due to the limitations of using purely logic-based knowledge representation techniques. Its main strength is its ability to express *uncertainty* in quantitative terms, i.e. to express to what degree something is certain (rather than, for example, first-order logic, which only allows expressing this through disjunctions like "either x or y is true"). Using this approach, causal relationships become statements about conditional (in)dependence and statistical correlation.

Bayesian statistics provides the mathematical foundation for computing the probability that a certain statement is true, by applying Bayes' rule for conditional probabilities. Assume Herbie is a self-driving, autonomous vehicle that can change location without a driver. We observe that Herbie is in London (fact A), and want to know how likely it is that Alice is there, too (fact B). Given our observation, this probability can be computed by considering the likelihood that both A *and* B are true (how often are they both in London?) as a proportion of the *prior* probability that Herbie is in London (how often is Herbie generally in London?). Formally, this can be written as follows:

$$\underbrace{P(\mathtt{at}(\mathtt{Alice}, \mathtt{London}) | \mathtt{at}(\mathtt{Herbie}, \mathtt{London}))}_{\text{conditional probability}} = \frac{\overbrace{P(\mathtt{at}(\mathtt{Alice}, \mathtt{London} \land \mathtt{at}(\mathtt{Herbie}, \mathtt{London}))}^{\text{joint probability}}}{\underbrace{P(\mathtt{at}(\mathtt{Herbie}, \mathtt{London}))}_{\text{prior probability}}}$$

Assume Herbie has been spotted 20 times in London (this is the prior probability, before we include new evidence that Herbie is there now), and Alice and Herbie were seen there 5 times.[6] If we observe that Herbie is in London, this would lead us to believe that Alice is there with a probability of $5 \div 20 = 25\%$. If, out of a long series of observations, we estimated the *prior* probability of Alice being in London (regardless of any statement about Herbie's status) to be also 25%, then we would have that $P(\texttt{at}(\texttt{Alice}, \texttt{London})|\texttt{at}(\texttt{Herbie}, \texttt{London})) = P(\texttt{at}(\texttt{Alice}, \texttt{London}))$, i.e. the locations of the two would be *conditionally independent*.[7]

This very simple example only scratches the surface of a whole range of the kinds of uncertain knowledge that can be expressed using *graphical models* [29] that describe complex probability distributions governing the behaviour of large numbers of variables. Probabilistic methods have resulted in a broad range of effective algorithms for performing queries on such models, and have been proven superior to "binary" logical models that are usually too brittle to adequately express real-world domains adequately. This is particularly true of applications where sensors and actuators have only limited precision or may occasionally malfunction, such as robotics or speech recognition. But it also reflects the ability of quantitative uncertainty to deal with certain aspects of the frame problem, as quantiative uncertainty allows us to make simple statements about "all that is known to be unknown" without having to explicitly list all factors and events that may lead to unexpected behaviour of the system. For example, a statement like "there is a 5% probability the car will break down" allows us to estimate how likely an action is going to fail while making an aggregate statement about the nature of all factors that could lead to such failure.

The key tradeoff that has to be made in return for this robustness is that computationally tractable probabilistic methods restrict themselves (roughly speaking) to the level of propositional logic. This means that each state and action variable in the system has to be treated individually, and we cannot easily process more abstract, generalised rules that make statements over entire classes of objects. In terms of design knowledge, use of probabilities implies that accurate quantitative knowledge about the base probabilities of all facts and the correlations between them have to be entered in the system, which requires much more fine-grained assumptions than simply knowing what is "true" or "false" in a logic-based system.

Looking at these methods from an anticipation point of view, it is important to point out that they can be used to express diagnostic, causal, or predictive knowledge, so they do not always have an anticipatory, forward-looking element. When they do, each of them adds different types of complexity in terms of design-time anticipation to the model of a problem environment compared to the techniques we have discussed previously.

In the case of logic-based reasoning, fairly compact sets of assumptions about the world can lead to complex domain theories, allowing very large (potentially infinite) numbers of statements to be inferred at runtime. On the other hand, as these theories become more complex, it becomes harder to anticipate the outcome of queries to a knowledge base, or the amount of time it will take for a query to return a result. As before, this illustrates that the more anticipation is enabled on the side of the system – here in the shape of inference of

[6] This is a simplified account of the meaning of probabilities following a frequentist model that estimates the probability of an event based on its past frequency over a long period of time. While other interpretations of probabilities exist, this one is sufficient for our example.

[7] Note that this does not preclude the events being causally linked, as the two probabilities might be coincidentally identical despite them being linked. One should not confuse correlation with causation, and in fact much more complex theories have been proposed to model causation in a probabilistic setting [36].

implicit, not directly observable properties of the world – the less the human designer can anticipate solutions generated (and thus, potentially, decisions influenced) by this system.

In the case of probabilistic reasoning, the novel element this introduces is making explicit how much about the world is known in quantitative terms. In terms of anticipatory capability this is a real step change compared to our previous models, as it implies that no statement is ever considered "true" or "false", but just becomes more or less likely. In other words, all alternatives always remain within the realm of possibility, and are simply weighted by relevance based on available evidence. An additional benefit of this approach is that the system can also quantify how *certain* it is about its estimations or predictions, which is interesting from the standpoint of anticipation, as it endows the system with a reflexive capability that allows it to gauge how much it can rely on its predictions. On the other hand, the amount of knowledge that has to go into an accurate model of a domain governed by uncertainty, and the computation that may have to be performed to update the probabilities of relevant variables in order to answer a concrete query, however, demonstrate the demands on the designers of systems who aim for this level of anticipatory capabilities in a system.

3.4 Learning

The previous section introduced methods that allow designers to add knowledge and observations to an AI system using general-purpose languages that can be effectively manipulated by computational procedures, but not for the system *itself* to extract this knowledge from observation. Methods developed in the area of *machine learning* [2, 33] aim to enable AI systems to construct models of the world from observed data, and roughly fall into three areas, which we will briefly introduce in the following sections.

In *supervised learning*, data is presented to the algorithm in the form of *input-output samples* and the task is to predict the output on input previously not seen. As an example, consider a set of journeys given as a list

$$\texttt{Edinburgh} \to \texttt{Glasgow} \to \texttt{Manchester} \to \texttt{Nottingham} : \mathit{Bad}$$
$$\texttt{Edinburgh} \to \texttt{Carlisle} \to \texttt{Leeds} \to \texttt{York} : \mathit{Good}$$
$$\texttt{Aberdeen} \to \texttt{Edinburgh} \to \texttt{Glasgow} : \mathit{Bad}$$
$$\texttt{Glasgow} \to \texttt{Newcastle} \to \texttt{York} \to \texttt{Sheffield} : \mathit{Good}$$

where the route contains features of the input, and the output (good/bad) indicates whether Alice enjoyed the journey (and is here just a single feature, but this is not a necessary restriction). The task of a learning algorithm is to derive a model for this data that would allow it to accurately predict whether Alice would like an arbitrary journey, e.g. whether $\texttt{Edinburgh} \to \texttt{Newcastle} \to \texttt{Glasgow} \to \texttt{Inverness}$ would be classified as *Good* or *Bad*. Since the possible outputs can be viewed as "classes" of items, supervised learning is often called *classification*, and the notion of "supervision" derives from the fact that training data used to choose a hypothesis (here about what makes a good or bad journey) contains "ground truth", ostensibly provided by a human or real observation. In this sense, the learning process is supervised – there is a source of reliable information that tells the algorithm which learning samples belong to which category. The hypothesis output by a learning algorithm based on the data in our example will be determined by looking for patterns and regularities in the training data. In our example, a sensible hypothesis might be that Alice only likes a journey if it passes through York.

Many algorithms have been proposed for performing such learning tasks, which are often based on some form of *adaptation* of an initial hypothesis based on every data item seen. *Decision tree learning*, for example, builds a tree that makes a series of decisions by checking the value of each input attribute at internal nodes, with the leaf nodes of the tree labelled by output values. It considers how different values of the input attributes predict each other (e.g. the value of the first city in the journey vs. the second, the second vs. the third, and so on), and is able to ignore irrelevant features altogether.

Neural networks emulate simplified networks of brain cells, where a neuron "fires" once the sum of the outputs of its preceding neurons exceeds a threshold and propagates its own value of activation (normally a number between 0 and 1) to successor neurons. The result of a new data sample is obtained by feeding the activation of input features through the network and reading off the predicted output at a designated output layer. In our example the output layer might contain just a single neuron, indicating that "good" is a value above 0.5 and "bad" below 0.5. In a nutshell, training a neural network involves evaluating the error produced on a new sample given the current weights between connected neurons that determine how activation is propagated, and adjusting these weights to minimise this error. Using techniques like neural networks one can learn arbitrarily complex numerical relationships between variables, though it is hard to "see" the final hypothesis by examining the network (it is implicitly encoded in the values of the numerical weights between the neurons), yet it is easy to input an unseen sample and obtain the predicted output.

Bayesian classifiers model the correlations between features in the data explicitly as joint probability distributions between them, updating these distributions as they track the frequency of co-ocurrence of specific variable values during training. This is achieved by applying the principles discussed in section 3.3.

Many other similar methods have been proposed in the literature for supervised learning, but what they all have in common is that they apply what is called an *inductive bias* to hone in on regularities in the data early, and have a facility of evaluating the quality of their current hypothesis by way of some form of built-in *critic*, usually based on prediction error. Inductive bias, which can be implemented, for instance, by preferring simpler hypotheses over more complex ones, is essential to the success of machine learning algorithms. If such an algorithm were to consider all hypotheses equally likely in each step, it would be unable to detect frequently observed patterns in the data soon enough, which would slow down the learning process [33]. At the same time, excessive bias may lead to *overfitting*, i.e. deciding to "lock into" a subset of hypotheses before having seen a sufficient amount of data that is representative of the real-world phenomenon the algorithm is trying to model.

These problems are equally important in *unsupervised learning*, where data is not labelled with explicit and reliable output features to be predicted, and the algorithm is instead trying to detect regularities mostly by looking at shared patterns in the data. An important class of these algorithms are *clustering* techniques [26], which attempt to group different data items together based on similarities between them. This process often involves checking which attributes of the data samples lead to the most coherent clusters, where coherence is often based on having as few clusters as possible while making sure samples are as close to each other based on the distance metric applied, and/or having as few outliers that are far away from any cluster centre as possible. Unsupervised learning is often used as an initial stage before coming up with a clear idea of what to predict based on the data, e.g. when data samples contain hundreds or thousands of variables, and it is unclear which of these are relevant. Many advanced machine learning algorithms involve a process of *feature selection*,

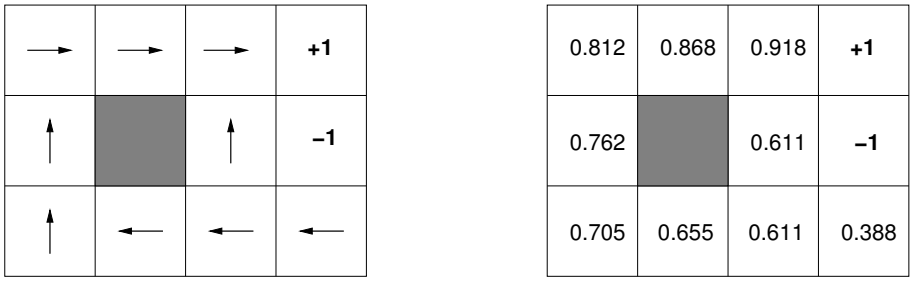

Fig. 3. A typical reinforcement learning problem: The agent has to navigate a grid, but when moving forward, may with some probability turn left or right involuntarily. There are only two states where a reward of +1 or -1 is received. The optimal policy is indicated by the arrows in figure (a), and is derived from the estimated utilities shown in figure (b). It indicates that in the bottom row of the grid, for example, choosing the longer route to reach the high-reward state is better than risking the low-reward state.

which attempts to compare the predictive value of different features to simplify the learning process by ignoring features that are less predictive and just constitute "noise" in the data.

The third category of learning methods, *reinforcement learning* [56], can be seen as a form of *semi-supervised learning*. Here, the algorithm is not given precise information as to which examples are positive and negative (or belong to which output class, in a more general setting), but a numerical reward that provides some relative feedback for the decision of the algorithm. Most commonly, reinforcement learning is applied in a setting of sequential decision making. The system experiences a current world state, and may choose to perform an action from a defined set of alternatives, which will stochastically lead to a new state, and produce a numerical (stochastic) *reward* informing the system of the quality of its decision in this individual step. This feedback may be delayed, for example when driving around various cities, the user may only tell the system that she liked the journey at the very end of the trip, and there may be a notion of terminal states, which, when reached, conclude an individual *episode* of training that counts as a complete training sample.

In reinforcement learning, the goal of the system is to come up with an *optimal policy*, i.e. a mapping from states to actions that will tell it what choice to make in each step, after a number of training episodes. In this context, a policy is considered optimal if it maximises the expected total reward achieved along the way among all possible alternative policies. If every transition from a state to its successor states only depends on the current action (i.e. previously taken actions do not matter for the current step), the world can be represented as a so-called *Markov Decision Process* (MDP) [40]. An MDP is essentially a large probabilistic search graph with rewards attached to some (or all) nodes, and as such represents a generalisation of search-based problem solving with elements of probabilistic methods. Figure 3 shows an example of a reinforcement learning problem adapted from [45].

If transition probabilities and rewards are known and do not have to be approximated from trial and error at runtime, a reinforcement learning problem degenerates into a *decision-theoretic planning* problem [3]. This can be viewed as a generalisation of a planning problem as defined in section 3.2 with additional non-determinism and a *graded* notion of goal achievement, whereby the total aggregate reward is maximised across all visited states, rather than just reaching a set of goal states.

If the problem is only *partially observable*, the AI system only has uncertain information about which state it is in at every step, and has to additionally maintain hypotheses about which observations map to which states. For example, our self-driving car may not know which city it is currently passing through and has to estimate its location from weather information. This is known as a *Partially Observable Markov Decision Process* [28], viewed by many as the most general formulation of an AI task (sometimes called "AI normal form"), as it involves a stochastically behaving dynamic environment with partial observability and uncertain outcomes of actions.

Many methods used in reinforcement learning, such as *Q-learning* [60], proceed by maintaining a running estimate of the quality of certain actions of states which is propagated from successor states to predecessor states periodically based on the idea that a state (and action) is worth visiting if the states that can be reached from it produce high rewards (shown in part (b) of figure 3). Through repeated trial and error, these algorithms are able to converge to the optimal policy in arbitrary MDPs under certain conditions.

These conditions mainly require that, even though the environment is non-deterministic, the probability distributions that govern this non-determinism do not change over time, and that the right balance is struck between *exploration* (trying out new alternatives to avoid missing opportunities to achieve rewards higher than those experienced in the past) and *exploitation* (repeating decisions that have produced good rewards in the past). Typically, such a balance will involve high degrees of exploration in early stages of learning, and increasing exploitation in later stages in order to converge to a *stable* policy in the long term. The exploration-exploitation problem is an example of the broader issue of making sure a machine learning algorithm applies the correct *sampling strategy* whenever it can influence the choice of what training samples to consider.

Learning algorithms mark a clear step change in terms of anticipation, both for the designer and for the system. At a meta-theoretical, epistemological level, they are based on the fundamental assumption that a correct model of the future behaviour of the world can be derived from observation of its past behaviour, i.e. that the future is an extrapolation of the past. Of course, we know of applications, for example predicting stock markets, where even human experts have been often faced with situations where no existing model predicted the experienced future behaviour, so this assumption does not always hold true.

In machine learning, the main mode of anticipation is one of *data-driven prediction*, and places much more weight on empirical evidence than the previous methods, which mainly relied on a designer-driven modelling process. From the designer's point of view, this implies that the coupling between their view of the world and that of the AI system becomes looser when using machine learning. Even though the designer can still make certain choices, e.g. which training data to supply to the learning system, how many training iterations to perform, which "critic" method to apply, and how to fine-tune the parameters of the algorithm, she cannot anticipate what the *output* of the algorithm will be. Hence, even though many of the *properties* of the algorithm can be verified offline (e.g. through mathematical analysis), its runtime *behaviour* cannot be fully anticipated.

Moreover, the inspection (and thus, intervention) opportunities some of the learning methods provide determine to what extent a human designer or operator can identify (and modify) what the system has learned. For example, neural networks or reinforcement learning algorithms are often encoded in large matrices of numerical weights that are hard to understand for a human. This means that even if the system is behaving as anticipated for some time, there is no guarantee that it will do so in the future. Such problems may occur even after the

learning phase is complete and the system is simply applying what it has learned so far, but they are even more likely if it will keep adapting its hypothesis in the future.

From the point of view of the algorithm itself, its anticipatory capabilities will depend on the accuracy of the model learned, which is largely determined by whether enough data has been provided to the system that is representative of the long-term behaviour of the task environment. Generally speaking, for any realistically complex process it is impossible to know whether enough data has been seen, though this can often be estimated by looking at how well the system is performing on new, unseen data. It is important to point out that despite the power of machine learning methods, effective use of learning-based methods remains challenging in complex, real-world tasks (as the AlphaGo [51] example of section 1 shows). In practice, it often requires a handcrafted choice and integration of different methods (in the case of AlphaGo, randomised search, neural networks, and reinforcement learning) with human-based testing and iterative fine-tuning to achieve good performance. To some extent, this is a consequence of our currently lacking abilities in terms of anticipating the performance of a learning system, but it remains to be seen how much of this process itself can be automated in the future, in order to enable AI systems to *learn how to learn*.

4 Anticipation in intelligent, autonomous agents

So far, we have only considered AI methods that are run *offline*, i.e. executed by a human designer or operator in order to obtain a solution to a given problem. At least since the mid-1990s, the notion of an intelligent and autonomous *agent* [63] has received much attention, which extends this offline view by a view of AI that has a piece of software and/or hardware operating persistently in an environment to achieve some task on behalf of its designer. An agent perceives its environment through *sensors* (in a very broad sense, from physical sensing devices to keyboard input or a message received over a computer network), and acts on this environment through *actuators* or *effectors* (again, interpreted in a very broad sense). This *situatedness* in an environment (that can usually only be partially controlled, and may contain other agents, each with their own spheres of influence [50, 61]) marks a departure from the traditional view of computation as the process of mapping some input to some output in a one-off, self-contained procedure, to a view of *autonomous* operation in an environment [9, 34, 44]. This shift from a program to an agent has major implications for anticipation.

One way of looking at this is considering the *horizon* of anticipation at the time of deploying the system into its environment. If we could perfectly anticipate all possible future circumstances while a solution returned by the system is executed, we could essentially come up with a *lifelong* plan that would determine the system's operation until it terminates or is decommissioned. This, however, would require a complete understanding of all details of the problem domain, which is unrealistic for most real-world tasks. Therefore it is more realistic to think of system designs that only consider a limited time horizon, attempt to make the best decision up to this limit, implement this decision, and then repeat this procedure. In heuristic search, this is often necessary because the search tree is too large to be explored completely (cf. the Go example of section 1), and the system only looks ahead a limited number of rounds. In planning, real-world systems often use *execution monitoring* which may lead to *re-planning*, and may even occur before a plan has failed or has reached its (limited) time horizon, simply to check whether circumstances are not as anticipated at the time of calculating the original plan [12]. In learning, the agent may re-start its learning procedure

from scratch every once in a while to avoid putting too much emphasis on regularities in the data that occurred much earlier on and might now be obsolete.

Another perspective we can take is that of responding to changes in a dynamic environment, which requires a balance of *reactivity* and *proactiveness* [27]. Proactive behaviour is based on anticipating future circumstances based on information about the current state of affairs, and is obviously essential to taking *goal-rational* action directed at achieving some objective. In itself, it is not capable of dealing with a dynamic environment where things may change, and this capability is generally believed to also be an essential part of human and animal intelligence that we should replicate in AI systems. If Alice has started her trip from London to Edinburgh via York, it would be unreasonable to stick to this plan if the car breaks down at York. A more sensible strategy might be to call a garage, have the car repaired, and then resume her journey. Of course, a more provident Alice might have considered this before departing, and her plan might have included this contingency as part of her overall policy, but this would require, at least in the limit, the kind of lifelong planning described above.

A third view closely related to the previous two, is that of *bounded rationality*. This concept, pioneered by Simon [52], and then mapped onto a concrete decision-theoretic mathematical model by Russell [47], involves considering the *resources* available for reasoning when making decisions. In its simplest form, bounded rationality states that rationality is limited by the amount of information available, the limited capabilities of any reasoning process, and the amount of time available to make a decision. Limiting the number of future states to be explored in an anticipatory process, or only processing as much training data as can be analysed within a certain fixed amount of time are typical examples of applying bounded rationality, but there have been also notable attempts to make the AI system reason about how much effort it should put into reasoning. Reasoning about how to reason, also known as *meta-reasoning* [48, 49], is based on the following idea: Assume you have information about how solution quality depends on effort spent to come up with the solution, and an idea of how much the reasoning process itself costs per possible alternative considered. Once the cost of reasoning outweighs the possible additional gain by performing more reasoning, it is not worth engaging in further reasoning.

Suppose Alice needs to stop for a meal on her trip from Edinburgh to London, and she is on a tight budget, so she would like to choose the cheapest restaurants on the motorway. She has access to a restaurant app on her phone, and can, in principle, look up any restaurant, visit its web site and compare prices, but this is a tedious process that takes time. Quite likely, picking a random restaurant might result in a high bill, but after comparing several restaurants, the additional gain will start diminishing, as it is unlikely that further options will be much cheaper. Assume considering a new alternative costs £1 to Alice, and the expected gains in every step (from first to second option, second to third option etc) follow the pattern £5, £2, £1, £0.5, £0.1 and are 0 thereafter. After one step, she will find a solution that makes her save £5, but will have spent £1 thinking about it, i.e. she obtains a total gain of £4. In the second step, this number is £4 + £2 - £1 = £5, in the third $£5 + £1 - £1$, and thereafter the additional costs outweigh further benefits. This means she can stop considering alternatives after having checked two options.

Such meta-reasoning of course requires additional knowledge about how useful additional reasoning is in a given problem domain, but whenever this is the case, it enables a different type of anticipation – one that involves a certain degree of introspection and awareness of the agent's reasoning capabilities.

4.1 Autonomy and intelligence

All three aspects discussed above ultimately relate to the same crucial property of agents, *autonomy*, as they all involve "letting the agent loose" (even if only for a while) and allowing it to make decisions at runtime, rather than having worked out a full solution to the problem in advance and having the AI system simply implement it. Despite the fact that autonomy is such a key aspect of agency, and transfers anticipatory power from the designer to the artificial agent, it remains one of those most extensively debated in the AI community [61].

The range of definitions proposed for autonomy usually includes the following: an agent is autonomous if it is able to operate without external intervention (this often closely associated with physical detachment from a human operator, e.g. in autonomous robots); the more the agent's behaviour depends on its own experience, the more autonomous it can be considered to be (in the sense that it exhibits behaviour that was not precisely anticipated at design time, this is particularly relevant if the agent is learning from experience); the less an observer knows about an agent's internal functioning or is able to predict its behaviour, the more autonomous the agent is (this emphasises interaction with another agent/human and external insight and understanding); the agent is the more autonomous, the less it obeys external commands (this emphasises self-determination, and is closely related to the risk of AI systems "running wild").

The relationship between autonomy and *intelligence* is also a tricky one. Commonsense intuition has it that the less guidance we have to give to someone, the more complex the tasks we can delegate to them reliably, the more variation in their behaviour we observe, the more flexible they are in responding to different circumstances, the more knowledge they are able to extract from their own experience to improve, and the more they are able to rationally pursue their objectives, the more intelligent they are. While these properties are all undoubtedly linked to different aspects of autonomy, we can also find many examples where autonomy exists without intelligence and vice versa: A thermostat is a very simple device that is completely autonomous (except for the user's intervention in specifying the desired temperature), but hardly intelligent. A pocket calculator can perform mathematical operations that are far beyond the cognitive abilities of a human, yet has no autonomy at all.

We believe that the concept of anticipation may be helpful in clarifying things at this point. If a human designer can anticipate the behaviour of the system, whether by analysing its behaviour through extensive testing, by mathematically proving the properties of the algorithm it implements, or by making assumptions about the circumstances under which the system will operate, then no matter how complex the actual computations carried out by this system or the behaviours it generates, it has no *genuine* autonomy in the narrower sense, and can be operated in a predictable way. If a system has anticipatory processes *itself*, then part of this human anticipation will have been delegated to it deliberately, and it will appear to exhibit (at least some degree of) autonomy.

4.2 Telling an agent what to do

Given these considerations, the final question to consider is how, given increasing amounts of autonomy in agents, we can influence and anticipate their operation at design time. If we want these agents to perform complex activities on our behalf, this boils down to answering the question "*how should we tell an agent what to do?*"

In section 3, we have already seen how objectives can be encoded into different types of AI algorithms: in search-based problem solving, we only distinguish between solution states and

non-solution states, i.e. the distinction between success and failure is binary. Planning maintains this distinction by considering goal states and non-goal states, but uses representations that allow us to inspect whether a goal has been partially achieved. Knowledge representation techniques consider a solution as the answer to a query, establishing whether it is true or false, or how likely it is to be true. Learning techniques aim to maximise prediction accuracy or to optimise long-term rewards obtained when acting in an environment, adding a further refinement to the definition of objectives by introducing a quantiative notion of success.

Hence, these techniques afford us with different ways of specifying objectives by way of defining which states should be achieved (or avoided), what questions should be answered, or that a certain degree of "utility" representing a performance metric should be maximised.

When moving from a monolithic view of a single execution of an algorithm to the notion of an agent that repeatedly engages in making decisions based on the current situation using some form of anticipation, how can we adapt our methods for specifying these objectives appropriately for them to be applicable in this setting? To answer this question, it is worth considering different types of *agent architectures* [61], models for the internal structure of agents that specify what meta-reasoning control mechanism is used on top of the internal problem-solving algorithms they use.

In our bounded rationality example above, we have already implicitly introduced one very general architecture, that of *decision-theoretic agents* (which is also implemented by the reinforcement learning agents as described in section 3.4). Decision theory [17], developed largely in the behavioural sciences and economics, postulates that a rational choice in a decision-making situation is one that *maximises expected utility*: Given the probabilities of all possible outcomes of a decision, and the utility of each of these outcomes, we should choose the decision that maximises the average "return" we expect from this decision, taking the uncertainty into account that is reflected by the probabilities of outcomes it may lead to.

This view of rational behaviour is based on a mathematically rigorous formalisation of benefit and uncertainty, it can be applied over any time horizon the agent wishes to make decisions over, it can accommodate updates to the utility values and probabilities in the world model using the methods we have introduced above, and is generally applicable in any problem formulation as long as there is a defined state and action space. Also, these elements are mapped to numerical utilities that reflect satisfaction of the agent's design objectives. Using probabilistic methods and techniques such as reinforcement learning, algorithms have been proposed that enable provably optimal rational agent design [46]. This model, however, also requires that the human designer has a way of assigning utilities to every possible state that might be encountered, and it may require a large amount of exploration to converge to an optimal policy, or may even be intractable in realistic state spaces.[8]

At the opposite end of the spectrum between "pure", mathematically grounded, adaptive methods, we find *deliberative* architectures like the Beliefs-Desires-Intentions (BDI) model of rational agency [20]. These architectures stand in the tradition of logic and planning, and draw from models of human *practical reasoning* [6]. Roughly speaking, BDI models (and the programming languages that have been developed to implement such agents [41]) are based on the idea that the agent has general *desires*, i.e. preferences regarding things it would like to achieve, but which may only be achievable or relevant in different situations. For example, when Alice's car breaks down, fixing it will have higher priority than continuing her trip to

[8] For this reason, elements of this model are used in many real-world applications, but often combined with other, more *ad hoc* meta-level control components.

London. At different points in time, the agent considers its current *beliefs* about the world (which are revised in every step based on observations) to determine which desires are worth pursuing. This process is called *deliberation*, and while the agent is deliberating, no specific commitment is made to pursuing a particular goal. Once such a decision is made, the desire becomes a concrete *intention* that is a concrete goal the agent will remain committed to unless it becomes achieved or unachievable [11]. For example, if Alice commits to fixing the car, she will continue to make "reasonable attempts" to achieve this intention.

Additionally, *intention reconsideration* [62] rules may be supplied to the agent that will force it to deliberate again even though its current intention has not been achieved or become unachievable. In our example, such a rule might be "if you start feeling unwell, reconsider your intentions", and make her drop the intention to fix the car and adopt an intention to visit a doctor instead. While an intention is active, the agent will try to create a plan to achieve it (e.g. drive to a garage, speak to the mechanic, wait until the car is fixed, and pay the bill) using techniques such as automated planning, or simply retrieve a suitable, pre-fabricated plan from a *plan library*.

The BDI model essentially represents a meta-level architecture that acts like a continuous planning loop, trying to map high-level design objectives encoded as desires to concrete plans for action by generating new, concrete goals at different points in time [5]. It can thus be seen as a typical example of attempting to break down "lifelong" planning into smaller, manageable chunks in order to balance reactivity and proactiveness, and to implement a notion of bounded rationality by performing only a limited amount of planning in every deliberation cycle, so that responsive, real-time behaviour can be achieved.[9]

If every intention adoption rule is a simple stimulus-response pair saying "in situation X, perform action Y", and all Y are single-step, atomic actions that immediately succeed or fail (and thus the intention is always abandoned after a single step, and the agent deliberates again), these architectures degenerate to purely *reactive architectures* [14] such as the subsumption architecture [7] that involve no anticipatory planning, and simply embed a hierarchy of simple behaviours executed one by one based on the current state.

While BDI is largely based on a less mathematically rigorous model of rational agency than decision-theoretic methods, it embeds anticipatory processes that seem to resemble more those of human practical reasoning. It is therefore maybe not surprising that a number of programming languages have been designed that enable a designer to write down models of plans for specific intentions, belief conditions that trigger the adoption of intentions, and similar other rule-based elements to control an agent.

Arguably, with respect to anticipation, it may be the case that models like BDI provide a better "interface" between the anticipatory processes of humans and those of the AI systems they build, as they make it easier to structure and inspect the methods an artificial agent uses to anticipate different circumstances in the world and make appropriate decisions.

5 Anticipating the behaviour and impact of AI systems

Throughout this chapter, we have made several remarks on the implications the use of different AI techniques has for human anticipation regarding the behaviour of systems that use them. We described the *teleological* [30] underpinnings of investigating AI and implementing AI

[9] This also prevents the agent from taking so long to create complex plans that the world may have changed by the time it has computed its plan, and it may no longer be relevant.

systems, highlighted the shift from *prediction* of behaviour to *assumptions* about a system, which is more strongly pronounced the more reasoning and autonomous decision making is transferred to the AI system, and discussed how increasing *complexity* in the system and its environment reduces the capacity of a designer to make precise predictions.

Putting all these ideas together, we can attempt a more comprehensive characterisation of the anticipatory processes involved when humans develop and use AI systems. Viewing such anticipation in the light of theories such as *situated cognition* [55] and the *extended mind approach* [10], both the models embedded in AI systems and the predictions about their behaviour can be viewed as a tool for human problem solving and learning. But what are the mechanisms involved in this anticipatory process? To answer this question, we need to determine *how* and *when* anticipation occurs in this context, and what its *outcomes* are.

Generally speaking, there are three mechanisms that can be employed when anticipating AI behaviour: *mathematical analysis*, *observation*, and *inspection*. Whenever mathematical analysis is possible (e.g. a proof of the completeness of a search algorithm or of the convergence of a machine learning algorithm), we can establish firm results about the properties of the behaviour of the AI system, typically *before* its operation. How universally valid these results will be, and how much we trust them, will of course depend on the way the properties are formulated, including their generality (e.g. does the algorithm produce the desired behaviour on a specific problem, or on a whole range of problems?) and the assumptions we have made when conducting the analysis (e.g. do the properties we are verifying accurately reflect what we want to know?). Clearly, as analytical methods operate in the abstract realm of formal models, their inherent reductionism will always limit the ways in which they can capture the full complexity of the real world. Whenever AI systems *themselves* use formal proof methods (as, e.g., in knowledge representation and reasoning), this limitation equally applies to them. In fact, this criticism has motivated a whole line of AI research that rejects such methods [8].

Observation of an AI system, on the other hand, occurs either in simulation, which allows human designers to assess and correct its behaviour without risking negative impact in the real-world application domain, or when they are deployed in a real-world environment to perform a task. On the positive side, observation is almost always *possible*, and can help assess a much broader range of behavioural properties of a system, including those hard to formalise using mathematical tools (e.g. is a robot nurse acceptable to its elderly users?). But like any empirical method of analysis, its anticipatory strength relies on the assumption that past observations are representative of the future. There are examples of domains where human history shows that the future does not necessarily extend the past, such as unpredictable financial crises, or complex – so-called "wicked" – decision-making situations that lack clear precedent (e.g. hard societal problems like climate change).

In the case of AI systems that learn from experience, an additional problem is that their future behaviour cannot necessarily be extrapolated from past behaviour. Numerous results, for example on the *fairness* of machine learning algorithms that generate predictions on population data and may embed unwanted, discriminatory biases [18] show that it can be very hard, if not impossible, to preclude any such adverse effects when the future output of an AI system is contingent on the future inputs it will experience.

While mathematical analysis and observation can be employed on other kinds of artefacts beyond AI, inspection of the internal models of an AI system offers much richer mechanisms than those available for other technologies. In terms of "looking under the hood" of the system, techniques that use human-interpretable internal representations are likely to afford us with more opportunities to anticipate their behaviour. We have already commented on

how different AI representations and agent architectures vary in this regard, but it is worth pointing out that in the current research and policy landscape, much effort is being invested in developing *explainable* AI, as witnessed, e.g., by DARPA's major research initiative in this area [23]. An important feature of AI in this respect is that it provides methods that go significantly beyond the traditional notion of "inspection", as AI systems can *interact* with humans at runtime, and this affords us with a broader range of options to diagnose, instruct, and correct their behaviour. There is much work, for example, on robots learning from demonstration [1], and recently the issue of AI systems learning to align their values from observation of humans [53] has become a hot topic in the area. From the standpoint of anticipation, such methods imply that anticipating the behaviour of AI must involve anticipating our own behaviour toward them, and much more work is needed to develop a deeper understanding of this issue.

There is one important issue that the above discussion neglects, namely that of *emergence* [21, 24]. In the tradition of reactive/behavioural AI [7, 8, 14], which views intelligence itself as a phenomenon that emerges from the interaction of an agent with its environment, there have been many approaches to develop AI systems by using bio-inspired and evolutionary approaches [16], or considering the collective intelligence that emerges from the simple interactions between large numbers of agents [57] and/or humans [31]. At first glance, it would seem that emergence, and with it, emergent intelligence, is conceptually rather at odds with the concept of anticipation as far as humans anticipating the behaviour of AI systems is concerned. After all, if the intelligent behaviour emerges, this means that it was not anticipated at design time. But this is, we believe, a somewhat oversimplified view.

Consider, as an example, the famous Mars rover simulation [54], where a robot trying to collect rock samples drops crumbs from a sample it is carrying that can be sensed by other robots, thus directing them to areas of the planet's surface where more samples are likely to be found. Experiments with this system show that, even though the individual robots only follow very simple stimulus-response rules and cannot directly communicate with each other, this is sufficient for complex global patterns of behaviour to emerge that could be considered intelligent. While these patterns could not be precisely predicted, the designers of the system certainly built the system to test whether they would emerge, and had the objective for the robot collective to be able to perform the task effectively. So some slightly more elaborate process of anticipation occurred, one which involves some anticipation of emergence itself. The designers hypothesised that simple rules governing the behaviour of a simple robot (e.g. change direction if you encounter an obstacle, drop a crumb if you are carrying a sample, etc) would together give rise to effective individual behaviour, and that placing several identical robots in the same environment would yield effective collective behaviour.

Yet, ultimately, this kind of anticipation of emergence is rather uncertain, and it is hard to analyse in terms of the typical errors that can occur in the process, i.e. having bad models, bad effectors, or bad side-effects [30]. This is because the observed behaviour cannot be directly causally linked to the model of the AI system the designer assumes, or to the models of the world the AI system itself uses.

6 Conclusions

In this chapter, we have provided an overview of anticipation in core AI techniques and in the design of intelligent agents. To this end, we introduced fundamental concepts and methods used in mainstream AI at a non-technical level, while aiming to convey as concrete as possible a sense for the capabilities of modern AI technology. For each of the techniques discussed, we

attempted an analysis of the types of anticipatory processes it embeds, and what consequences this has for the ways in which human designers and users of AI systems can anticipate their behaviour.

An assumption that underlies our treatment of the topic, and which we have tried to argue for throughout this chapter, is that the more anticipation is left to an AI system, the more we are limiting the ways in which humans will be able to anticipate its behaviour. This tension may prove to be ultimately unresolvable if we want to endow these systems with capabilities that emulate (or even surpass) human intelligence – after all, we expect these intelligent machines to encapsulate enough flexibility and providence to deal with situations we cannot anticipate at design time.

Crucially related to this is also the insight that the more complex a problem is, the harder it becomes to specify what exactly we want an AI system to achieve. In fact, many of the discussions surrounding the risks of AI highlight the issue of the objectives of future intelligent systems being misaligned with those of humans, but again, this may be a fundamentally unresolvable issue. Driving a car safely, winning a game of Go, performing complex surgery – fundamentally no human can give another human concrete, explicit instructions on how to achieve that in a comprehensive way, and while we rely on teaching and providing advice, the "learner" of these tasks has to fill in the gaps with their own reasoning resources and experience over time through trial and error.

Although this may be a rather speculative conclusion, we believe that, to balance anticipation among humans and intelligent systems in practice, appropriate techniques are needed that allow for inspection of and interaction with complex AI systems. In particular, it seems like explanation and justification of decisions would allow humans to better understand what models of anticipation artificial agents apply when making them, a research issue that has been largely overlooked in the literature. Also, the design of future AI systems needs to afford users facilities to communicate with these systems in such a way that they can rectify not only errors these systems make, but also the design errors humans are responsible for due to a lack of anticipating future circumstances when designing these systems.

References

1. B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.
2. C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.
3. C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
4. R. J. Brachman, H. J. Levesque, and R. Reiter. *Knowledge Representation*. MIT Press, 1992.
5. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.
6. M.E. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
7. R. A. Brooks. Intelligence without representation. *Aritificial Intelligence*, 47:139–159, 1990.
8. R.A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.
9. C. Castelfranchi. Guarantees for Autonomy in Cognitive Agent Architecture. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Proceedings of the First International Workshop on Agent Theories, Architectures and Languages (ATAL-94)*, pages 56–70. Springer-Verlag, 1995.
10. A. Clark and D. J. Chalmers. The extended mind. *Analysis*, 58:7–19, 1998.
11. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
12. M. desJardins, E. H. Durfee, C. L. Ortiz Jr., and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

13. S. Edelkamp and S. Schroedl. *Heuristic search: theory and applications.* Elsevier, 2011.
14. J. Ferber. Reactive distributed artificial intelligence: Principles and applications. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 287–314. John Wiley & Sons, New York, NY, 1996.
15. R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.
16. D. Floreano and C. Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies.* MIT Press, 2008.
17. S. French. *Decision Theory: An Introduction to the Mathematics of Rationality.* Halsted Press, 1986.
18. S. A. Friedler, C. Scheidegger, and S. Venkatasubramanian. On the (im)possibility of fairness. *CoRR*, abs/1609.07236, 2016.
19. D. M. Gabbay, J. H. Siekmann, and J. Woods, editors. *Handbook of the History of Logic, Volume 9: Computational Logic.* Elsevier, 2014.
20. M. P. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge. The belief-desire-intention model of agency. In J. Müller, M. P. Singh, and A. S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 1–10. Springer-Verlag: Heidelberg, Germany, 1999.
21. C. Gershenson and N. Fernández. Complexity and information: Measuring emergence, self-organization, and homeostasis at multiple scales. *Complexity*, 18(2):29–44, 2012.
22. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice.* Morgan Kaufmann, 2004.
23. D. Gunning. Explainable artificial intelligence (xai), 2017.
24. J. H. Holland. *Emergence: From Chaos to Order.* Helix Books, New York, 1998.
25. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
26. Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data.* Prentice-Hall, Upper Saddle River, NJ, 1988.
27. C.M. Jonker and J. Treur. Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness. In W.P. de Roever, H. Langmaack, and A. Pnueli, editors, *Proceedings of the International Workshop on Compositionality (COMPOS-97)*, Lecture Notes in Artificial Intelligence vol. 1536, pages 350–380. Springer-Verlag, 1998.
28. L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.
29. D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.
30. A. H. Louie. Robert rosen's anticipatory systems. *Foresight*, 12(3):18 – 29, 2010.
31. T. W. Malone, R. Laubacher, and C. Dellarocas. The collective intelligence genome. *Sloan Management Review*, 51(3):21–31, 2010.
32. J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
33. T. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.
34. M. Nickles, M. Rovatsos, and G. Weiss, editors. *Agents and Computational Autonomy - Potentials, Risks and Solutions. Postproceedings of the First International Workshop (AUTONOMY 2003), July 14, 2003, Melbourne, Australia*, volume 2969 of *Lecture Notes in Computer Science.* Springer-Verlag, 2004. 275 pages.
35. Dimitri Ognibene, Eris Chinellato, Miguel Sarabia, and Yiannis Demiris. Contextual action recognition and target localization with an active allocation of attention on a humanoid robot. *Bioinspiration & Biomimetics*, 8(3):035002, 2013.
36. J. Pearl. *Causality.* Cambridge University Press, 2009.
37. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Francisco, CA, 1988.
38. R. Poli. An introduction to the ontology of anticipation. *Futures*, 42(7):769–776, 2010.
39. R. Poli. The many aspects of anticipation. *Foresight – The journal of future studies, strategic thinking and policy*, 12(3):7–17, 2010.
40. M. L. Puterman. *Markov Decision Problems.* John Wiley & Sons, New York, NY, 1994.
41. A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. van der Velde and J. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)*, Lecture Notes in Artificial Intelligence vol. 1038, pages 42–55. Springer-Verlag, 1996.

42. A. Riegler. The role of anticipation in cognition. In D. M. Dubois, editor, *Computing anticipatory systems*, volume 573, pages 534–541. Proceedings of the American Institute of Physics, 2001.
43. R. Rosen. *Anticipatory Systems: Philosophical, Mathematical, and Methodological Foundations*. Pergamon Press, 1985.
44. M. Rovatsos and G. Weiß. Autonomous Software. In S. K. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering. Volume 3: Recent Advances*, pages 63–84. World Scientific Publishing, River Edge, NJ, 2005.
45. S. J. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach*. Pearson Education (Prentice-Hall), Upper Saddle River, NJ, 2 edition, 2003.
46. S. J. Russell and D. Subramanian. Provably Bounded-Optimal Agents. *Journal of Artificial Intelligence Research*, 2:595–609, 1995.
47. S.J. Russell and E.H. Wefald. *Do the right thing: Studies in limited rationality*. The MIT Press, Cambridge, MA, 1991.
48. S.J. Russell and E.H. Wefald. Principles of rationality. *Artificial Intelligence*, 49(1-3):361–395, 1991.
49. Stuart J. Russell and Eric Wefald. Principles of metareasoning. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *KR'89: Principles of Knowledge Representation and Reasoning*, pages 400–411. Morgan Kaufmann, San Mateo, California, 1989.
50. Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems – Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.
51. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
52. H. A. Simon. *Models of Bounded Rationality: Empirically Grounded Economic Reason*. MIT press, 1982.
53. N. Soares. The value learning problem. In *Ethics for Artificial Intelligence Workshop at 25th International Joint Conference on Artificial Intelligence (IJCAI-2016) New York, NY, USA 915 July*, 2016.
54. L. Steels. Cooperation between distributed agents through self-organization. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I.*, pages 175–196. North-Holland, Amsterdam et al., 1990.
55. L. A. Suchman. *Plans and situated actions: The problem of human-machine communication*. Cambridge University Press, New York, NY, 1987.
56. R.S. Sutton and A.G. Barto. *Reinforcement Learning. An Introduction*. The MIT Press/A Bradford Book, Cambridge, MA, 1998.
57. K. Tumer and D. H. Wolpert. A survey of collectives. In *Collectives and the Design of Complex Systems*, pages 1–42. Springer, 2004.
58. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Journal of Mathematics*, 58:345–363, 1936.
59. C. Vondrick, H. Pirsiavash, and A. Torralba. Anticipating visual representations from unlabeled video. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, pages 98–106, Las Vegas, June 26th to July 1st, 2016.
60. C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.
61. M. Wooldridge. *An Introduction to Multiagent Systems, 2nd edition*. John Wiley & Sons, Chichester, England, 2009.
62. M. Wooldridge and S. D. Parsons. Intention reconsideration reconsidered. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V*, volume 1555 of *LNAI*, pages 63– 80, Berlin, Germany, 1999. Springer-Verlag.
63. M. J. Wooldridge and N.R. Jennings. Agent theories, architectures, and languages: A survey. In M. J. Wooldridge and N.R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Artificial in Artificial Intelligence, vol. 890, pages 1–39. Springer-Verlag, Berlin et al., 1995.