

Planning as Modelling in AI

By Paul Almond, 26 November 2006

Website: www.paul-almond.com
Email: info@paul-almond.com

© Copyright Paul Almond, 2006. All Rights Reserved.

Planning As Modelling in AI

By Paul Almond, 26 November 2006

Introduction

In a previous article *How AI Would Work* [1], a general proposal for artificial intelligence (AI) was made, including a description of an approach to planning in AI in which planning is a special case of modelling. The *planning as modelling* approach was discussed in more detail in *Occam's Razor Part 9: Representation and Planning of Actions in Artificial Intelligence* [2].

The AI approach uses a probabilistic hierarchy of meanings and other aspects of the system, beyond planning, have been discussed in other articles [3,4,5,6,7].

Planning as modelling breaks the partition between planning and modelling which has traditionally been a feature of AI. This partition still applies in the hierarchical system of meaning extraction proposed by Jeff Hawkins [8,9] which features a special output hierarchy, “closely-coupled” with the meaning extraction hierarchy. Such partition of planning and modelling is redundant in the planning as modelling approach.

Planning as modelling means that the modelling system probabilistically predicts the AI system's future inputs *and outputs*. The modelling system attempts to predict both external events in the real world and its own behaviour, with no distinction between them. The AI system's *predictions* of its own future behaviour are equivalent to *planning* of its own future behaviour and can be used to constrain a search for optimum behaviour to such a degree that almost all of the work involved in such a search is already done by the application of the constraints. Planning therefore occurs almost completely within the modelling system as a trivial special case of modelling.

Planning as modelling has been described in previous articles in the context of the specific hierarchical, probabilistic modelling system [1,2,3,4,5,6,7] which has been proposed. The idea is more general than this and can be used with other probabilistic modelling systems. This article has two purposes:

- to describe planning as modelling in a more general context, separate from the specific details of the probabilistic hierarchy previously proposed, so that planning as modelling is easier to understand and not seen as dependent on other features of the AI system.
- to describe a change to the concept described in previous articles about planning [1,2] which will improve the AI system's planning ability by including inputs in its search for optimum behaviour.

This means that I will be assuming that we have a working modelling system which fulfils certain criteria and will not be describing how it works in any detail. This is not a naive omission: it is deliberate in this article.

Some of the text that follows covers similar ground to a previous article [2] which discussed planning as modelling in the specific context of a hierarchical model, so in parts of this article I have reused text from the previous article, editing it as appropriate for the change in context.

A Change to the Concept

This article will make a change to the ideas discussed previously, as follows:

In the previous articles on planning I described a system that I called the *output vectoring* system as performing a search of the possible future behaviour of the system and checking possible sequences of future outputs. While a system could have integrity and work like this, it would actually perform better if the output vectoring system were searching the set of possible future sequences of outputs *and inputs* and I should have incorporated this into the design.

This article corrects this. The output vectoring system's search process incorporates both inputs and outputs and is therefore a search of possible futures, in which the machine's behaviour plays a role, rather than just a search of possible future behaviour.

Where relevant in the article I will comment on this again to make it clear where a design change has been made.

Requirements for the Modelling System

General Idea

Planning requires the AI system to have a modelling system. Unlike in previous articles I will not be describing how this works in detail. Instead I will describe the requirements for it in terms of external behaviour.

The AI system receives inputs. We can presume that each input event consists of a binary digit (bit) – a “1” or “0” input – being received by the AI system at some instant of time. The AI system makes outputs. We can presume that each output event is a bit – a “1” or “0” output – being sent by the AI system to the external world at some instant of time.

The modelling system analyses the history of input and output events. Each time an input or output event occurs we can imagine the input or output event bit being added to a record of all the input and output events that have occurred so far. The modelling system analyses this data and makes probabilistic predictions of future input and output events. In other words, by looking at the inputs and outputs that the AI system has received and made in the past, the modelling system makes probabilistic predictions of the inputs and outputs that the AI system will receive and make in the future.

Saying that these predictions are probabilistic means they have uncertainty. The modelling system determines the degree of uncertainty: it assesses its confidence in its

own predictions. For each input event or output event in which the AI system will receive or send out a “1” or “0” the modelling system produces a probability of that input or output being “1”, which also of course indicates the probability that the input or output is “0”. When the modelling system is confident in its predictions it produces probabilities for input events and output events which are close to 0 or 1. When it is very uncertain about its predictions it produces probability values which are close to 0.5.

No Distinction Between Inputs and Outputs

This sort of modelling system does not differentiate between inputs and outputs. The AI system’s outputs are fed into the modelling system as just another type of input. The modelling system does not even need to “know” which of the “inputs” being provided to it are actually the AI system’s own outputs being returned to it so it can observe them.

This lack of distinction does not actually need to be allowed for in the design of the modelling system. Provided that the modelling system works in the correct probabilistic way as described above, it can be made to generate predictions of input and output events simply by feeding both of them into it as if they are inputs. This requirement is therefore about how the modelling system is set up to interact with the rest of the AI system rather than about how it is constructed.

Graph Analogy

The role of the modelling system can be viewed in terms of graphs. Each of the AI system’s inputs or outputs at any instant is “1” or “0” and can be viewed as a graph showing the variation in its value (vertical axis) over time (horizontal axis). Graphs like this are common in digital electronics textbooks. Each position on the horizontal time axis would correspond to an input or output event. The time axis would actually be quantized so that any finite period of time contains a finite number of input or output events, but this need hardly bother us at this stage. The past input and output events of the system are stored as historical data (or some abstraction of these is stored, as will be discussed later in this article). With the graph analogy this means that the system stores all the graphs showing the previous behaviour of all of its inputs and outputs (or some abstraction of them) and there is no distinction between inputs and outputs: they are all treated as inputs.

The purpose of the modelling system is to predict what these graphs will be like in the future. Such predictions cannot be certain and must be expressed probabilistically. Instead of “1” or “0” values for the future graphs, probabilities are given. At any instant of time in the future there is a probability value indicating how likely it is that the graph for the particular input or output will be “1” at that instant. This also gives the probability that the graph will be “0” at that point: it is simply 1 minus the probability value.

Theoretically the modelling system produces an infinite number of probability values for any graph, each describing the expected future behaviour of an input or output at some instant of time in the future. In practice time is quantized. For each input or output graph

the time axis is divided into discrete input or output events and there is a single value of “0” or “1” on the graph. For any future projections produced by the system, there is a single probability value describing future expectation for each input or output event.

Wallpaper Analogy

Another way of viewing the modelling system’s role is in terms of “wallpaper”. This is an extension of the graph analogy in which all the graphs of input and output values are combined to make a single dot matrix pattern. We can imagine this pattern on a roll of wallpaper. The purpose of the modelling system is to extend this “wallpaper” pattern into the future. Uncertainty means the modelling system must express it as a matrix of probability values, each indicating the chance that the “wallpaper” should be marked at a particular location.

No Dependence on a Particular Modelling System

In previous articles I described how such a modelling system could work as a hierarchy involving partial models, meaning extraction, probabilistic interpretation of partial models and downward transfer of probabilities [1,3,4,5,6]. None of this matters in this article, which is not about how the modelling system works but about how a probabilistic modelling system can be used to perform planning.

This is an important concept in this article: the planning as modelling approach does not need a modelling system with any particular internal workings.

The Relationship Between History and Prediction

The modelling system relates the history of input and output events to predictions of input and output events. There are two ways in which it may be convenient to view this relationship and they amount to the same thing:

Input and Output History and Prediction Collection

The system’s modelling is based on the *input and output history* – the record of the input and output events that have occurred so far and produces the *prediction collection*. The prediction collection is the set of probabilistic predictions of future inputs and outputs made by the modelling system and contains probabilities for future inputs and outputs.

Using the graph and wallpaper analogies made previously, the prediction collection is the probabilistic extension of the “graphs” or “wallpaper pattern”.

This view involves describing future input and output events in the prediction collection in terms of probabilities, while the history of previous input and output events involves a definite value of “0” or “1” for each bit. The prediction collection is entirely determined by the input and output history and the modelling system is a practical implementation of the relationship between the input and output history and the prediction collection.

History and Predictions Represented as Probabilities

Instead of expressing the future predictions probabilistically and the history of input and output events as definite “0” and “1” values, as described above, it may make more sense to use a single structure to express both the history and predictions. A definite input or output event in the history can still be expressed as a probability – just one that has complete certainty associated with it. An input or output event that occurred in the past with a value of “1” is known certainly to have a value of “1” and so can be represented by a probability of 1. An input or output event that occurred in the past with a value of “0” is known certainly not to have had a value of “1” and so can be represented by a probability of 0. All past input or output events can therefore be represented by probabilities of 0 or 1.

If probabilities can be used to represent both past and future input or output events, a single entity can be used to represent both. Instead of having the input and output history and the prediction collection there can just be one set of probabilities describing both the past input and output events with values of 0 or 1 and future input and output events with varying probabilities. The idea of such an entity was used in the previous articles describing the hierarchical system [1,2], where it was referred to as the “bottom level” of the hierarchy. In this article we are not assuming that it is the bottom level of anything or that there is even a hierarchy in the modelling system. We will refer to it as the *combined probability collection*. We are only interested in it here as the point of contact between the modelling system and the rest of the AI system, and how this relates to everything else inside the model does not concern us.

In this article I will be discussing the functioning of the output vectoring system, and its interaction with the model in some detail and I will need to refer to the process of “fixing” of input and output bits. I will do this by describing it in terms of both the input and output history and prediction collection and the combined probability collection.

Updating of Predictions and Fixing of Bits

Any future input or output event bit is due to occur at some specific time in the future which, as time passes, will get closer to the present.

Any input or output event bit in the future will eventually make the transition from having uncertainty and being described by a probability to being in the past, when it is known about with certainty and described by a value of “0” or “1” or a probability of 0 or 1. The process of an input or output event bit’s value becoming known with certainty in this way will be known as *fixing*. All input or output event bits are fixed as they enter the present.

After being fixed an input or output event bit cannot change further. Assigning it an absolute value or probability admits no possibility that it could take any other value and as it is now in the past we clearly know what its value is.

The historical, fixed bits are related to the future probability values by the modelling system.

In the case of the input and output history and prediction collection, the prediction collection at any time is completely determined by the input and output history and the modelling system. Changes to the input and output history as bits become fixed therefore affect the prediction collection.

In the case of the combined probability collection things are the same: that part of the combined probability collection corresponding to future predictions of input and output events is completely determined by that part of the combined probability collection corresponding to past input and output events (and containing only probabilities of 0 and 1) and the modelling system.

The following should be noted:

- It is easiest to imagine the modelling system working just from historical data, so that it continually examines the input and output history or that part of the combined probability collection corresponding to the past and updates the prediction collection or that part of the combined probability collection corresponding to future predictions. It is not required that the modelling system is actually implemented like this – just that it is equivalent to it. The modelling system could base some of its future predictions on other predictions: as these would all be based ultimately on past data it would actually still be equivalent to using the history of input and output events as the basis for all prediction.
- If the modelling system is implemented so as to explicitly base predictions on the history of input and output events only then the fixing of an input or output event bit is significant: the bit abruptly goes from being something about which predictions are made to something that is used to make predictions. If, alternatively, the modelling system is not explicitly restricted to basing predictions on historical data then the fixing of an input or output event bit is less significant: using the combined probability collection view, for example, the bit's probability would simply get closer to 0 or 1 as the time of its occurrence got closer to the present and it would become increasingly used in the making of future predictions as this occurred.
- It is easiest to think of the history of input and output events as being explicitly stored within the modelling system but this is unnecessary. The modelling system could achieve something similar by storing some abstraction of some or all of the input and output history or that part of the combined probability collection which relates to historical events. We do not need to be concerned about this. We just need to be able to treat the input and output history and prediction collection or the combined probability collection as the interface between the modelling system and the rest of the AI system and we can treat the modelling system as if it is explicitly storing and processing its internal information in this way, regardless of whether or not any abstraction is occurring.

How Planning Works

The General Idea

Planning as modelling uses probabilistic predictions of future input and output events of the AI system to constrain a search for optimum outputs.

The search for optimum outputs is performed by a separate system, the *output vectoring system* [1,2,7]. The output vectoring system only determines those outputs that should be made now. Its search is constrained by the probabilistic predictions of future inputs and outputs.

This constraint reduces the amount of computation that the output vectoring system needs to do in planning, trivializing its role. Almost all the computation needed for planning is done in the production of the constraint – in making the predictions of future input and output events. Planning as modelling reduces planning to a trivial special case of modelling.

How Planning Would Work Without Constraint

To see how constraint can reduce the computation in planning, we will first consider how the output vectoring system would need to work without it. It is here that the change which I said would be made in this article – the incorporation of inputs into the search by the output vectoring system – will be introduced.

The output vectoring system tries possible sequences of inputs and outputs (in chronological order) and assesses the desirability of the consequences. After hypothetically making an output or receiving an input the modelling system could be used to simulate the consequences probabilistically by processing the input or output as if it had actually happened. The desirability of the outcome could then be assessed by a *situational evaluation function* – an algorithm which examines the probability values for the future inputs and returns a numerical value indicating desirability. Evaluation functions are already used in other algorithms, such as chess playing algorithms [10].

It would be inadequate just to try making those outputs that are going to occur now. The making of outputs affects the situation that arises later and the scope for improving it by making later outputs. It is therefore better to search through the possible sequences of future inputs and outputs to investigate the scope for later action, and the resulting desirability that can be achieved, for any output that can be made now. This would allow the output vectoring system to determine those outputs which can be made now that turn out to be the most desirable in the long term.

The output vectoring system does not need to search every possible sequence of future inputs and outputs. One way of avoiding the need for exhaustive searching is to use a tree search which allows various search tree “pruning” methods to be applied. This is an approach that will be discussed in more detail later in this article. A method like this does

not eliminate high demand on computing resources. It is impossible to search a collection of infinitely long sequences of inputs and outputs so this kind of tree search typically “cuts off” at some depth, which may be variable. Tree searching is used in other systems, including chess programs [11,12]. Even ignoring the benefits of pruning, a search tree is still the most obvious way to organize this kind of search.

How Constraint Can Reduce the Amount of Computation in Planning

The future predictions of inputs and outputs can be used to constrain the output vectoring system’s search, increasing its efficiency.

The use of constraint from a modelling system can be viewed in simple terms as the search for optimum behaviour proceeding through the “space” of possible future behaviour, with most emphasis on the future behaviour of the AI system that is considered most likely by its modelling system.

Constraint can reduce the amount of computation needed by the output vectoring system by not treating all sequences of inputs and outputs equally, instead giving different levels of importance – or priority – to different sequences of inputs and outputs, the amount of priority given being proportional to the probability of that particular sequence of inputs and outputs being experienced and made by the AI system in the future. As in the absence of constraint, the set of sequences of inputs and outputs is not searched in its entirety, but the set of possible future sequences of inputs and outputs that are examined is as large as possible, given limits on time and computing resources. The priority of any sequence of future inputs and outputs is used to determine whether or not it is included in this set.

How Probabilistic Predictions Can Be Used to Prioritize a Sequence of Outputs

Sequences of inputs outputs are prioritized according to probability.

For any sequence of input events and output events the probability of each individual input event or output event occurring can be obtained from the prediction collection or combined probability collection. The probability of an output of “1” being made or an input of “1” being received in the future is simply the probability value stored for the relevant input event or output event in the prediction collection or combined probability collection. The probability of an output of “0” being made or an input of “0” being received it is 1 minus this probability.

The overall probability of a sequence of inputs and outputs occurring is obtained by multiplying all the probabilities of the individual input and output events together.

Example:

Suppose the following sequence of five input events and output events is being considered:

11010

We do not need to be concerned with what these inputs and outputs are, or which are inputs and which are outputs: they are simply a possibility for the next five inputs or outputs that an AI system will receive or make, in chronological order.

The probabilities corresponding to each of these input events and output events in the prediction collection or combined probability collection produced by the modelling system are as follows:

0.75, 0.20, 0.56, 0.85, 0.1

The probabilities of the inputs and outputs in the sequence occurring are as follows:

0.75, 0.20, 1-0.56, 0.85, 1-0.1

The overall probability for the sequence is given by:

Probability=0.75x0.20x(1-0.56)x0.85x(1-0.1)=0.050

The higher this probability then the higher the priority assigned to this sequence of inputs and outputs and the more likely that it will be considered by an output vectoring system with limited resources.

The Need for Simulation

Simulation is needed by the modelling system. When considering a sequence of inputs and outputs the modelling system should be used to determine what the prediction collection or combined probability collection would be after this sequence of inputs and outputs occurred. This is necessary to allow the situational evaluation function to determine the desirability of the situation described by the prediction collection or combined probability collection after the inputs and outputs have occurred.

Simulation is also needed because occurrences of inputs and outputs affect the probabilities of later input or output events in the prediction collection or combined probability collection. The probability values should not just be looked up in the prediction collection all at once, before applying a sequence of inputs and outputs. Once each hypothetical input or output is assumed to have occurred, it should be known to the modelling system – instead of just being a probability value in the prediction collection – and the modelling system should be able to use this information to update the model and

(hypothetically) change the probability values for the other input and event output events. The actual or hypothetical occurrence of an input or output (by including it in some sequence of input events and output events) should therefore cause the modelling system to update the prediction collection or combined probability collection. When considering a sequence of inputs and outputs the modelling system should be used to simulate the effects of the occurrence of each, in chronological order, on the prediction collection or the combined probability collection. The probability value in the prediction collection or combined probability collection for each input or output, then, should be obtained after using the modelling system to take the effects of occurrence of the previous inputs and outputs in the sequence into account.

Use of a Tree Search

Prioritization of sequences of inputs and outputs as described above can improve a search, but this does not explain the mechanics of how a search process can use this prioritization to “concentrate” on certain sequences. If we only deal with whole sequences then it is hard to see how any prioritization could work: by the time we have considered a complete sequence of inputs and outputs, and whether its priority is high enough, it is too late: we have already spent computational resources on considering it.

The search process is most able to exploit prioritization if priority can be assessed for *groups* of sequences of inputs and outputs, which can then be omitted from consideration if their priorities are insufficient, without the need to consider each individually. This can be done by considering the priorities of *partial* sequences of inputs and outputs, finding those which are too low in priority and omitting all the complete sequences of inputs and outputs which include them, without needing to consider each of these complete sequences of inputs and outputs individually. This fits with what tree searches do.

The way in which probabilities are multiplied for consecutive events fits in with this. Multiplying probabilities makes the resultant probability smaller, so if there is a sequence of inputs and outputs for which the combined probability is already below some cut-off point, then any sequence of inputs and outputs containing it will have a still lower probability. We only need to examine the small sequence of inputs and outputs to determine that the priority would be too low for any of a number of longer sequences of inputs and outputs of which it is a part.

A tree search can be used to investigate the consequences of making outputs, while ensuring that only sequences of inputs and outputs with sufficient priority are investigated. A “cut-off” probability would be defined. The tree search would investigate the different inputs and outputs that could occur in chronological order. The tree search would only follow a new path from a node provided that that the priority for the sequence of outputs so far – obtained by multiplying the probabilities for all the individual input events or output events at the nodes above it – is at least the cut-off probability. This means the tree would be searched to a greater depth when likely inputs and outputs are involved.

One way of looking at this is that the tree search would experiment with possible sequences of inputs and outputs, much as a chess algorithm experiments with possible sequences of moves, to determine those outputs that are best to make *now*, with more attention – in the form of deeper searching – being given to those paths through the search tree which represent more likely behaviour by the AI system and external reality, so that such paths are explored to a greater depth.

An Example of How Planning as Modelling Could Work Using a Tree Search

I will now provide a more detailed description of how planning as modelling could work. This example will use a tree search as I briefly described above.

How the Output Vectoring Process Works

The output vectoring process is a recursive tree search similar to that used in chess algorithms [11,12]. It deals with input and output event bit probabilities in “absolute” terms: whereas a bit in the prediction collection or combined probability collection can have any probability value the output vectoring process works by assigning absolute values of “0” or “1” or probability values of 0 or 1 to input or output event bits.

A recursive tree search fixes some future input and output event bits: it puts a number of bits in the prediction collection into the history of inputs and outputs, assigning them absolute values of “0” or “1”, or looking at it the other way, assigns absolute probability values of 0 (meaning that the bit certainly has the value “0”) or 1 (meaning that the bit certainly has the value “1”) to bits in the combined probability collection. We do the assignment in chronological order, starting with the bits for which input or output is actually going to occur first; that is to say the input and output bits corresponding to the earliest input and output events.

For each bit that is being fixed in this way we fix it with both possibilities. If we are using the input and output history and prediction collection view this means that we assign the bit both possible values of “0” and “1”. In the combined probability collection view this means that when the bit is assigned an absolute probability we assign it both possible absolute probabilities – 0 and 1 (equivalent to assigning the bit the values of “0” and “1” respectively).

Each of these two possibilities corresponds to a separate branch in the search tree to a new node. For each of these nodes we consider the next input or output in chronological order and assign the relevant input or output bit both possible absolute values (“0” and “1”) or both possible absolute probabilities (0 and 1), each of these corresponding to a branch from the current node to a new node, before considering the next bit at each of these new nodes and so on. We are checking every possible permutation of assignments of values of “0” and “1” or probabilities of 0 and 1 to input and output bits.

Whenever we assign a value or probability to an input or output bit we allow the modelling system to respond by propagating the changes through the model to simulate

them, updating the remaining probability values in the prediction collection or the combined probability collection.

We cannot continue indefinitely: the number of possible paths through the search tree is 2^n , where n is the number of input and output bits being considered. When the search tree has reached a certain depth (when we have considered a certain number of bits) we therefore terminate it. This fixed-depth method of termination typifies the search tree as a flat search tree and is used in basic chess algorithms.

We have a number of possible terminations of the search tree, each involving a specific permutation of assignment of values of “0” and “1” or probabilities of 0 and 1 to bits corresponding to future input and output events. At each such termination we determine the desirability of the situation based on the score associated with the probabilities in the prediction collection or combined probability collection for bits corresponding to future inputs.

We want to select outputs which have high desirability scores, but we are not really choosing an entire sequence of actions: we only need to decide what to do *now*. The scores obtained at the terminal nodes of the search tree are “backed up”: they are passed back up to the start of the search tree, in a process somewhat similar (but not identical) to the “backing up” of scores in chess search trees.

Backing Up of Scores

“Backing up” is the process by which scores determined at lower nodes in the search tree are sent back up the search tree to higher nodes. The search tree branches – each node has two nodes immediately below it (except in the case of terminal nodes) – so the “backing up” of scores to a higher node means that information about the scores on two nodes must be sent back up to a single node where only one score is needed. There must therefore be a decision about what score to “back up” to a higher node.

The decisions are made differently depending on whether we are considering an input or output event.

Output events are dealt with as follows:

Where we have two branches from a node corresponding to an output event (for “0” and “1” as bit values or 0 and 1 as probabilities on inputs or outputs), each leading to a sub-node with a score, we place the *higher* of the two sub-node scores on that node. This node, with its score, is then considered as one of the two sub-nodes of whichever input or output it followed on from, and so on.

Input events can be dealt with in one of two obvious ways as follows:

Where we have two branches from a node corresponding to an output event (for “0” and “1” as bit values or 0 and 1 as probabilities on inputs or outputs), each leading to a sub-

node with a score, we place the *mean (expected) score* or the *lower* of the two sub-node scores on that node. This node, with its score, is then considered as one of the two sub-nodes of whichever input or output it followed on from, and so on.

The mean (expected) score is not a simple numerical average of the two sub-node scores, but the expected value of the score, taken into account the probabilities of occurrence of each score as frequencies. The probabilities for each input occurring are obtained from the value stored in the prediction collection or combined probability collection for that input event.

As an example:

Suppose we have a probability of 0.62 stored in the prediction collection or combined probability collection for an input event. The score previously backed up to the “1” node for this event is 25 and the score previously backed up to the “0” node is 30. We want to know the score to “back up” to the node above.

If we have decided to back up the mean (expected score) then the value backed up is:
 $((0.62)(25)+(1-0.62)(30))/2 = 13.45$

If we have decided just to back up the lower score then the value backed up is the lower of 25 and 30, which is 25.

After repeated backing up of scores like this, we eventually end up with scores for assigning values of “0” and “1” or probabilities of 0 and 1 to the very first output bit considered in the search tree. We make the output – “0” or “1” corresponding to the higher of these two scores – and fix the relevant output event bit at “0” or “1” by putting the bit in the input and output history with a value of “0” or “1” or assigning it a probability of 0 or 1 in the combined probability collection.

The computation is then repeated for the next output that is required.

Scores resulting from inputs and outputs are dealt with differently because the AI system has a different degree of control in each case.

In the case of an output the system can be optimistic – choosing the higher of the two scores – because it can select such an outcome simply by making the relevant output.

In the case of an input the system needs to be more cautious. It does not have control over which way things will go and which node will become relevant. There are two obvious approaches: backing up the mean (expected) score or being more pessimistic and backing up the lower of the two scores. Backing up the mean has the advantage of describing the situation without any prejudice: the score placed on the higher node is simply the average score that is expected to result. Some readers may think that this is too dangerous and prefer to back up the lower score and other readers may prefer some compromise between these two approaches that I am not considering here.

Backing up the lower of the two scores may appeal to some readers because of its use in chess. When thinking about this kind of system I find it helpful to imagine it playing chess. The outputs are analogous to moves made by the AI system while the inputs are analogous with moves made by its opponent. With a chess program the system has to assume the worst case for the opponent's moves because the opponent is "malicious". An AI system like this may need to deal with intelligent agents in the world that oppose it and will make sure that the worst possible outcomes result for it, but this does not in itself mean that the lower score should be chosen. We did not just take the average of the two scores, but calculates their mean, taking into account the *probability* of each. These probabilities should already allow for the risk that something in reality "conspiring" to select low scoring outcomes will do so and this feature is missing from the standard chess search tree.

Using the low score, instead of the mean (computed with regard to probabilities), does not necessarily mean assuming that malicious agents are at work in the environment. Such an assumption would just be a special case. This sort of pessimistic view is more likely to be taken when it is more important that the AI system meets minimum standards of behaviour, in terms of avoiding situations with particularly low scores, rather than simply maximizing its average score. This may be of relevance, for example, where safety issues are involved.

There could even be situations in which the output vectoring system is made to take the *higher* of the two scores. This could happen in applications where we want the AI situation to achieve some outcome, but are not bothered by a lot of low scores as long as it sometimes does it. The AI system may, for example, be built to achieve an ambitious goal *once*.

The output vectoring system of a working AI system is unlikely to work at such extremes as choosing the higher or lower of two scores backed up to inputs. It is more likely that it would back up the mean (expected) score, but it may not do this. A better approach may be to use some compromise between the mean (expected score) and the lower score or (less likely in most situations, I think) between the mean (expected score) and the higher score.

What Has Been Changed

As I mentioned previously, a change has been introduced in the concept. This change is that inputs now feature in the output vectoring system's search as well as outputs. This change does not affect the main ideas – of planning being a special case of modelling and the modelling system constraining a search for optimum behaviour – but it will make the output vectoring system work better.

The output vectoring system is now more similar to a chess algorithm (though this could cause the misconception that it is actually the AI system's "planning system"), in that it examines possible future developments in terms of the system's own actions and things

that can happen in the outside world. This fits in much better with the idea of not trying distinguishing between inputs and outputs that is used in the modelling system.

It is useful to imagine this sort of system playing chess. The exclusion of inputs from the output vectoring system's consideration does not mean that the system, as previously proposed, would not work. In a previous discussion I took the view that this system, if playing chess, would automatically have its opponent's moves modelled as the possibilities would be described in the prediction collection or combined probability collection as probabilities that changed in response to the moves (outputs) made by the system. This view is still correct, but the problem is that it does not involve any modelling of the *particular* consequences that may follow a move. Although the consequences of actions are still represented as probabilities – everything with which we are dealing with here is – modelling inputs as well as outputs allows particular different situations that could follow from the occurrence of different inputs to be studied, rather than combining them all together to produce a vague probabilistic description of many possible futures. This will result in a much better system.

Using Predictions to Direct Behaviour

The output vectoring process described above is computationally intensive, but we can increase its efficiency. We can use the system's predictions of its own behaviour to make the search for optimum behaviour easier and I will now discuss this.

The modelling system produces the prediction collection – a set of probabilistically expressed predictions for future input and output events – or assigns probability values to bits in the combined probability collection. Prediction of the system's own outputs – meaning the modelling system is predicting the AI system's own behaviour – is automatic and needs no special mechanism: it results simply from feeding the system's own outputs into the modelling system along with the inputs.

The search process described above did not give any consideration to the likelihood of particular sequences of inputs and outputs occurring: it just analysed all possible sequences of inputs and outputs for a number of future input events and output events.

We can use these probabilities of future inputs and outputs to prioritize the search. We can consider the same number of possible sequences of inputs and outputs, but instead of including sequences in our consideration purely on the basis of number of input and output events (required search depth) we can include sequences in the consideration on the basis of *probability*.

To do this we run a planning tree search as described above, but changed as follows:

Before starting the tree search we define a value that I will call the *cut-off probability*. Sequences of future inputs and outputs with a probability of future occurrence of at least the cut-off probability will be included in the tree search and all other sequences of future inputs and outputs will be omitted. The choice of cut-off probability value is based on

how much computing we can do in the tree search. If we have a lot of computing power or a lot of time to do the computation then we choose a high cut-off probability, but if we have little computing power or available time then we choose a low cut-off probability. The minimum value of the cut-off probability is 0 – specifying consideration of an infinite number of sequences of inputs and outputs – and the maximum value is 1 – resulting in no sequences being considered at all.

We maintain a *running probability* value representing the probability of the particular sequence of inputs and outputs under consideration happening. The running probability starts with a value of 1.

Each time we go to a new node in the search tree (that is to say, each time we add a new input or output to the sequence) we multiply the running probability by the probability of the corresponding input or output value. If the input or output event bit at this node is to have a value of “1” or a probability of 1 then we multiply the running probability by the probability value of this input bit or output bit in prediction collection or the combined probability collection. If the input or output event bit at this node is to have a value of “0” or a probability of 0 then we multiply the running probability by 1 minus the probability value for this input or output in the prediction collection or combined probability collection. The resultant value becomes the new running probability.

After updating the running probability by the above operation at each node its value is examined. If the running probability is less than the cut-off probability then no further output bits are added to the sequence and the tree is terminated. This means that the search tree will get deeper when more likely paths through it are being considered while unlikely paths through the search tree will quickly reach the cut-off probability and be terminated.

It should be noted that this means that a sequence will be considered when its running probability is slightly less than the cut-off probability because the tree is terminated when this happens, rather than before it, but this does not matter much: when the running probability is less than the cut-off probability the search tree is not extended further and this adequately approximates what was specified earlier.

The tree search previously involved any sequence of inputs and outputs terminating when a certain number of inputs and outputs (or search depth) was reached. This feature of the tree search is removed. There is no fixed upper limit on the length of sequence that can be considered. All the limitation on the tree search is now dependent on the cut-off probability.

This search strategy is similar to the Shannon type-B search strategy [13] in chess programming. There is one difference: the Shannon type-B strategy is more about extending searches until quiescent positions occur, whereas this strategy is about extending them based on probability, but the same idea of selective extension of searches is in use.

What I have described here is the minimum needed for such a search to work, together with the most important constraint method. Other, more conventional ways of increasing the efficiency of the search will be available.

As I stated earlier, the tree search is a simulation of the effects of possible actions. This means that any changes to the running probability are also simulated and only apply below the node in which they are made. When we go back up the search tree to higher nodes they must be undone.

An Example of the Use of Predictions to Direct Behaviour

We will now look at an example of the use of probabilities on bottom-level output bits to constrain the output vectoring tree search.

Suppose we choose a *cut-off probability* of 0.017.

We start with a *running probability* of 1.

We look up details for the next input or output event to happen. This will correspond to a single input or output bit. Suppose its probability is 0.8, meaning there is a 0.8 chance that this bit has the value “1”.

We have two branches in the search tree. If we go down the “1” branch we will fix the value of this input or output bit at “1” or its probability at 1 and multiply the running probability by 0.8. If we go down the “0” branch we will fix the value of this input or output bit at “0” or its probability at 0 and multiply the running probability by $1-0.8=0.2$. We follow both branches, each with a different recursive call to the search process. We will just look at what happens with the recursive call that follows the “1” branch.

We follow the “1” branch of the search tree. The input or output bit under consideration is fixed with a value of “1” or a probability of 1. The running probability is multiplied by 0.8. $1 \times 0.8 = 0.8$ so the running probability is now 0.8. 0.8 is greater than the cut-off probability of 0.017 so the search can continue down this branch.

The next input or output bit to be considered has a probability of 0.72. There are two branches on the search tree. If we fix this bit with a value of “1” or a probability of 1 we multiply the cut-off probability by 0.72 and if we fix it with a value of “0” or a probability of 0 we multiply the cut-off probability by $1-0.72=0.28$. We follow both branches, each with a different recursive call to the search process. We will just look at what happens with the recursive call that follows the “0” branch.

We follow the “0” branch of the search tree. The input or output bit under consideration is fixed with a value of “0” or a probability of 0. The running probability is multiplied by 0.28. $0.8 \times 0.28 = 0.224$ so the running probability is now 0.224. 0.224 is greater than the cut-off probability of 0.017 so the search can continue down this branch.

The next input or output bit to be considered has a probability of 0.05. There are two branches on the search tree. If we fix this bit with a value of “1” or a probability of 1 we multiply the cut-off probability by 0.05 and if we fix it with a value of “0” or a probability of 0 we multiply the cut-off probability by $1-0.05=0.95$. We follow both branches, each with a different recursive call to the search process. We will just look at what happens with the recursive call that follows the “0” branch.

We follow the “1” branch of the search tree. The input or output bit under consideration is fixed with a value of “1” or a probability of 1. The running probability is multiplied by 0.05. $0.224 \times 0.05 = 0.0112$ so the running probability is now 0.0112. 0.0112 is less than the cut-off probability of 0.017 so the search can no longer continue down this branch.

This is now a terminal node of this search tree. The evaluation function examines the bottom-level input probabilities and assigns the search tree node a score which can be “backed-up” to earlier nodes as described previously.

Why Prediction Driven Behaviour Works

The system can start with no previous behaviour and learn to behave competently:

Initially there is no guidance from previous behaviour. The probability values in prediction collection or the part of the combined probability collection containing future predictions are all 0.5 and will not effectively constrain the tree search. The search process initially finds behaviour that is not very good, because of this lack of constraint, but it will still be slightly better than running no search process at all.

In later searches, this previous behaviour is used to constrain the search. The constraint works by means of the modelling system.

The previous inputs and outputs are passed into the modelling system and the prediction collection or combined probability collection is updated. The updated prediction collection or combined probability collection will include probability values for outputs that have been changed by the modelling system in response to the system’s recent outputs. The system’s recent behaviour (and in fact, all previous behaviour) will affect the modelling system’s predictions of its later behaviour.

It is these predictions, resulting from the *previous* behaviour, that are used to constrain the tree search finding the *current* behaviour. When the search tree was previously traversed there was no constraint at all, but the search would still have worked to some small degree. Now that this previous behaviour is constraining the search the constraint is better and the available computational resources can be used more efficiently to find behaviour better than what was previously found. This new behaviour will in turn form part of the constraint on future behaviour, allowing computational resources to be used to be used more efficiently to improve on it still further.

Planning Is Prediction

If the hierarchy contains predictions, and actually does most of the system's planning, then what I am saying here is that planning is really prediction. The modelling system is doing the planning.

This may seem strange to some readers, but it is natural. We often see people doing things and have a good idea – from modelling – what they are going to do next. Now, if we can apply modelling to work out what a person is going to do next, it follows that the person being modelled could also know what he/she is going to do next, given access to the same kind of modelling – and it makes a good case that this is how people determine what to do next.

This gives what may be a strange picture of the process of “making your mind up” about some decision. We are used to thinking of “making your mind up” as determining the optimum actions to make, given some model. In the context of this article it means something different. When you have not “made your mind up” about something you lack sufficient information to predict your actions with regard to it. When you experience the “making up of your mind” it means that you now have enough information in your model to predict what you are going to do.

Planning Need Not Be Simple

It may seem to some readers that using previous behaviour as a guide means the modelling system merely being expected to “copy” previous behaviour and generate future behaviour in a simplistic way. This is not the case. Basing future behaviour on previous behaviour means basing it on a *model* generated from past behaviour. The relationship between past and future outputs is merely that they are part of the same model: this model can have any degree of sophistication supported by available processing power.

Far from demanding that the system simply copy old behaviour, this actually allows it improve its behaviour. If the past behaviour shows a history of the system's performance in some task improving, and if there is enough of a history of inputs and outputs, then the most obvious model is one predicting further improvement for the system!

This may seem to some readers an attempt to get a “free lunch”. There is no “free lunch” in reality. Even with an inbuilt tendency to improve its own behaviour, the system is still dependent on how much abstraction the modelling system can provide. There will still be limitations on this, one of which is the available computing power.

Some readers may still ask how the system is supposed to “know” to improve itself, given that its behaviour is based on modelling from its previous behaviour. This “direction” to the system's behaviour is given by the output vectoring system, which does test possible outputs according to desirability. In a way, the output vectoring system can be considered a link between the modelling system (which is doing the real planning) and

the way that desirability of inputs is defined. I will now give a better idea of how this is supposed to work.

The Planning/Improvement Cycle

The way in which the system improves its behaviour can be thought of as the *planning/improvement cycle*. This cycle is not something that we need to do explicitly: it is just a simplified way of describing how the system's outputs can feed back into the model to control later outputs and the role that the planning system plays in this.

The system does not really get into the planning/improvement immediately, so the description below will include the leading up to the start of it.

1. The system initially has no previous inputs or outputs. All of the probabilities in the prediction collection or in the part of the combined probability collection relating to the future are 0.5. The output vectoring system achieves nothing and the system's behaviour is arbitrary.
2. Input and output events have now occurred and the modelling system contains patterns relating inputs to outputs. Any projection of future behaviour is based on arbitrary previous behaviour and is still useless. When the output vectoring system simulates inputs and outputs the modelling system can at least now alter the prediction collection or combined probability collection probabilities, because enough previous data exists to do this. This means that the output vectoring system should at least be able to assess the consequences of actions in terms of desirability. This is better than nothing and a slight improvement in behaviour should occur. The "planning" (in the output vectoring system itself), however, will be very short range, due to the lack of any effective constraint that is not arbitrary.
3. The system now has a model containing better behaviour from Step 2. The modelling system makes a projection based on this previous behaviour. This projection, expressed in the prediction collection or combined probability collection as output probability values, provides constraint on the output vectoring system's search. The modelling system itself is now having some influence on behaviour. There is no tendency in the modelling system to improve the behaviour (yet) but the output vectoring system will attempt to find the optimum behaviour, given the prediction collection or combined probability collection probability values. It manages to slightly improve on the previous behaviour in Step 2, because it has the same computational resources but now has the benefit of this constraint.
4. The system has previous behaviour that has been improving. This means that the model of its future behaviour will be one that describes an improving system. The probability values put into the prediction collection or combined probability collection by the modelling system will therefore constrain the output vectoring system in a way that describes improved behaviour – even before the output vectoring system starts working. By performing a tree search the output vectoring

- system is able to improve on this behaviour slightly, meaning that the improvement achieved now is better than the previous improvement.
5. The system's behaviour has not only improved in the past. Its *rate* of improvement has increased. The model therefore describes a system functioning in this way and the constraint in the predictions already describes a system which is improved more than last time. This constraint allows the system to improve more than last time, even without the output vectoring system doing any work. The output vectoring system does perform a tree search, however, and makes a very slight improvement on this. The rate of improvement has now been increased, and this will be reflected in future modelling of the system's behaviour by the modelling system without the output vectoring system having to provide it.

The planning/improvement cycle can be thought of as Step 5 repeating. There is a runaway aspect to this. Where I left it, the modelling system had not only acquired the tendency to improve the system's behaviour, but had the tendency to increase the rate of improvement of the system's behaviour.

Graphical View of the Planning/Improvement Cycle

The improvement need not stop with what was just described. If we imagine the desirability of the system's behaviour plotted on the vertical axis of a graph, with time on the horizontal axis, then the curve is gradually extended over time and the modelling system will automatically include any previous properties of the curve in the predictions of future behaviour which constrain the output vectoring system. This means that when the output vectoring system makes an improvement to the behaviour or the rate of improvement of behaviour, the model of the system's behaviour accounts for the improvement that has occurred so far. This has a good chance of meaning that the output vectoring system itself does not have to do anything to maintain the gradient of this curve, beyond preventing statistical drift, and is free to try to optimize the system's behaviour to make the curve steeper.

The "System X" Analogy

One way of thinking about how the approach described in this article works is as follows:

Imagine that you observe the behaviour of an artificially intelligent machine. You imagine that the machine is running some software called "System X" and you make a model to predict System X's behaviour. The machine itself, however, is doing the same thing. It is also making a model to predict the behaviour of "System X" and using it to *generate* its behaviour.

In a way, this could be taken as meaning that System X does not really exist! Even the computer which would be running System X is just modelling it and the mind, already arguably a virtual thing, can be viewed as a virtual shadow of itself. The artist, Rene Magritte's, work "La trahison des images" ("The treachery of images") comes to mind here.

People should not get carried away by what I have just said: it is intended more to aid understanding of what the system is doing than as a serious philosophical claim and it could be argued that System X exists by virtue of the very attempt to model its behaviour – that System X and the model of it are equivalent.

Scope of the Output Vectoring System

A possible objection to all this is that the sort of search performed by the planning system is too short-range, simple and lacking in abstraction to resemble what we perceive as planning of our own actions.

As an example, when a great military leader is planning the fall of an enemy empire he has some view of the “endgame” far in the future and what to do to achieve it. This view, however is abstract and a suggestion that it is described in terms as detailed as probabilities of future input and output events would be implausible.

The fallacy, here, would be in thinking that long range, abstract planning of actions occurs in the output vectoring system. The search process in the output vectoring system resembles planning, but it is not the system’s real planning process: that occurs in the modelling system itself, where greater abstraction is available. The output vectoring system is just there to establish a pattern of previous improvement and provide stability.

Prioritization Control Outputs

Why Prioritization is Needed

However the modelling system is constructed, the problem of prioritization of computation resources will need to be solved. The modelling system will need to decide what input data is most relevant in generating the probabilistic predictions of inputs and outputs. It will also need to decide which out of many possible computations are most relevant. Some computations will be based on intermediate results and the decision about which intermediate results to use could be complex. All of these decisions may vary from time to time and the modelling system needs to adapt the way it computes accordingly.

This can be summarized as follows:

The modelling system needs to think only about relevant things.

This issue of keeping a computational system “focused” like this is often known as the *carpet texture problem*.

Some way is needed of controlling prioritization of computational resources in the modelling system – what it concentrates on – at any time.

It should be noted that the word “prioritization” is being used in a different way here than earlier in the article. It now refers to the relative importance of different types of processing in the modelling system rather than the relative importance of different possible sequences of future inputs and outputs being examined by the output vectoring system as previously.

How Prioritization Control Outputs Work

Some of the AI system’s outputs are designated as special *prioritization control outputs*. Prioritization control outputs do not control events in the outside world. Instead, they control prioritization of computational resources within the modelling system.

Apart from what they control, prioritization control outputs are dealt with in the same way as other outputs:

- There is no special planning involving them – the existing planning system involving interaction of the modelling system and output vectoring system encompasses the prioritization control outputs.
- The prioritization control outputs are fed back into the modelling system along with the other conventional outputs as the inputs, so that the modelling system can use them in making its predictive model.

This means that the AI system is making outputs to control its own modelling system just as it would manipulate its outside environment. In a way, the modelling system *becomes* part of the outside environment, an idea which I call *AI as a boundary system* and is discussed in a previous article [7].

The details of how a given prioritization control output will affect prioritization in the modelling system will need to be resolved. These are dependent on specific details of the modelling system and beyond the scope of this article.

What prioritization control outputs do resolve is the issue of how the system learns how to adjust the internal prioritization in its modelling system. It learns how to do this in the same way that it learns how to plan any other aspect of its behaviour.

Prioritization control outputs that inappropriately set priorities in the modelling system will cause it to spend its limited computational resources wrongly and work in a sub-optimal way, with too much uncertainty in areas where more certainty was needed. This could happen for various reasons:

- All of the probabilistic predictions of inputs and outputs have a lot of uncertainty because processing has been wasted on computing intermediate results that have little affect on these predictions.
- Processing has been wasted on achieving an unnecessarily high standard of prediction for a small number of input and output events.

- Processing is being wasted on achieving a high standard of prediction for irrelevant input and output events.

If the system produces this sort of behaviour – and initially it will – the modelling system will be inefficient. The output vectoring system will encounter a lot of uncertainty in simulations of actions and predictions made by the system will not achieve high situational evaluation function scores. If, however, the system produces behaviour that involves better prioritization then the model predictions will have less uncertainty for those inputs and outputs where it matters and the system will be able to achieve high evaluation function scores.

This will affect the search process of the output vectoring system. When it tries prioritization control outputs that result in simulations by the modelling system with too much uncertainty to get good situational evaluation function scores, and also tries outputs that result in simulations with less uncertainty that do allow it to achieve good scores, the prioritization control outputs that reduce uncertainty in useful ways will be found to be better and will be selected by the output vectoring system. The output vectoring system does not need to know that it is doing this by using the prioritization control outputs: in fact, it does not need to know which of its outputs are prioritization control ones and which are conventional. In this way, the output vectoring system slightly “nudges” the prioritization control outputs made by the system in the direction of better modelling. Once made, these prioritization control outputs are part of the system’s behavioural history and will naturally play a part in the predictions of future behaviour made by the modelling system. In this way, the system will learn to organize itself. There is nothing special about this process: this is how the AI system learns any other type of behaviour.

Prioritization Control and Integrity

It is possible to make a fallacy here by asking how, if the prioritization control outputs are all wrong, as they must be when the AI system is started up, we can know that use of the modelling system in simulations will cause low scores? If the modelling system is not working properly, what would stop it wrongly making predictions giving high scores and suggesting that the incorrect prioritization control outputs are good? This fallacy would be based on the idea that the prioritization control outputs control everything in the modelling system. This is not the case. Prioritization control outputs do not affect the integrity of the modelling system at all. The integrity of the modelling system must always be assured irrespective of what the prioritization control outputs are. The prioritization control outputs do control how the modelling system spends its computational resources and which aspects of the modelling are done in detail and which are represented by abstraction.

Prioritization Control, Irreversibility and Forgetting

There is no reason, in principle, why prioritization outputs should not be able to make irreversible changes to the modelling system, provided that these do not interfere with its

integrity; for example, prioritization outputs could order some detailed information to be removed from the modelling system and replaced by abstraction.

This may be particularly relevant with regard to the issue of storage of the historical data about past input and output events in the modelling system. Earlier, I said that this data may or may not be stored explicitly. If the data is stored explicitly then the input and output history, or that part of the combined probability collection corresponding to past input and output events, is explicitly stored in the model, whereas if the data is not stored explicitly then some abstraction is stored. Explicit storage of all this historical data could require a lot of storage capacity and a greater problem may be that storage of this data will mean that it gets processed, potentially using a lot of the system's resources. I find it unlikely that the human brain explicitly stores all the historical data in this way and I would expect I am in agreement with most people on that. Such abstraction can be directed by prioritization control outputs which replace detailed historical data by abstracted versions of it when the benefits of abstraction in terms of reduced storage capacity requirements and processing outweigh any loss of accuracy in prediction. This would be *forgetting*. It would be valid for prioritization control outputs to do this provided that the integrity of the modelling system remained intact.

It should be noted that, if the modelling system is caused to “forget” parts of the historical record of input and output events like this, the prioritization control outputs causing it are planned within the modelling system itself: forgetting is not being imposed from outside, but rather the modelling system is itself determining what it needs to forget as part of the planning as modelling process.

Obtaining a “Better View”

If some readers have trouble with the idea of prioritization control outputs, and the reason that no special type of learning is needed for them to work, this analogy may help:

First, imagine a robot which uses planning as modelling. It is capable of planning actions in the world. This means manipulating the world to improve its situational evaluation function scores. Suppose there are obstacles, which could be easily moved, blocking the robot's view and what is behind them may be relevant to the robot's situation. We should not be surprised if the robot moves the obstacles. Doing so could reduce uncertainty in some aspect of its future predictions, allowing it to chart a path through these predictions that improves its score.

Suppose that after moving the obstacles the robot sees a computer scientist who offers to make some improvement to its modelling system. The robot should not need any special type of behaviour to evaluate this offer. If the scientist's claim is correct, accepting the offer would involve making outputs that result in the system having better probabilistic predictions – just like the decision to move the obstacles.

In both moving the obstacles and accepting the scientist's offer the AI system is simply making outputs that give it a “better view”. Whether this “better view” is achieved

through making outputs that just alter the environment or making outputs that alter the modelling system in ways that allow better scores to be achieved is irrelevant.

Instead of the scientist we can now imagine the robot finding a tool kit and making the alterations to its own modelling system by itself and we can take this further. Ultimately, we are left with certain outputs that directly affect prioritization within the model system.

This gives a simple way of viewing prioritization control outputs: as do-it-yourself brain surgery.

While specific details of how prioritization control outputs work within the modelling system need to be decided for any specific modelling system used, prioritization control outputs are the general way in which the carpet texture problem should be solved.

The Importance of Prioritization Control

Although the modelling system needs to ensure that the model has integrity without prioritization control, this does not mean that prioritization control has a minor role of “fine-tuning” a modelling system. Setting up the prioritization in a modelling system is a critical part of making the model itself.

As an example, the simplest modelling system with integrity could just spew out the value “0.5” for every prediction of a future input and output event. Such a system may have integrity, but there is nothing there. A more sophisticated modelling system lacking any kind of prioritization control may attempt to analyse everything. It would not get very far though: time constraints imposed by the need to act in the real world would limit such computation and the computation that did get done would be almost arbitrarily. By trying to analyse everything, such a modelling system would analyse almost nothing.

Without prioritization control, a modelling system lacks sophistication. Prioritization control provides this and is a deep part of what the modelling system is. An intelligent system is not just a modelling system, with some prioritization control helping it to think more efficiently. This would be like saying that the Darwinian evolution is “just about reproduction” and that random variation and natural selection “just make it work better”. Rather, the modelling system lacks significant sophistication and is like a featureless block of stone out of which prioritization control carves a mind.

Use of a Directed History

The system’s response to inputs is based on modelling of previous inputs and outputs. There needs to be a history of desirable behaviour to establish a pattern of desirable behaviour which modelling will continue or, which is better, a history of *improvement* in behaviour to establish a pattern of improvement in behaviour which modelling will continue.

The output vectoring system deals with this by finding ways to improve the system's predicted behaviour slightly, so that this new behaviour can be used instead and will then itself contribute to the pattern of previous behaviour. The output system therefore establishes the pattern of improvement in behaviour for modelling to continue.

An alternative way of establishing a pattern of desirable behaviour improvement in behaviour could be to set up an artificial pattern of previous behaviour in the input and output history or combined probability collection as a *directed history*. This could be done in a number of ways. One way would be to directly program the information. Another way would be for a human to control the unit for some time and "act" the part of a system which is exhibiting desirable behaviour or, preferably, an improvement in behaviour. I am unsure about how often this would be done: it is not really part of the main idea, but needed mentioning as a possibility. It would probably be done more often for simpler AI systems where a specific type of desirable behaviour, rather than any improvement, was required or for some more sophisticated AI systems with very specific behavioural requirements.

In an earlier, speculative article *Indirect Mind Uploading: Using AI to Avoid Staying Dead* [14] I discussed the concept of a computer system designed to model the behaviour of a human by observing his/her actions, or having access to records of them. Such a system would be very similar to a system using planning as modelling with a directed history.

Conclusion

Planning as modelling has been described in a way showing its independence of specific features of the hierarchical, probabilistic AI system described in previous articles [1,2,3,4,5,6,7]. A change to the concept has also been described. Previous descriptions of planning as modelling featured a search for optimum outputs, involving analysis of possible future outputs, constrained by the modelling system. This search will now also involve analysis of possible future *inputs*.

Planning as modelling uses probabilistic predictions of future input and output events of the AI system to constrain a search for optimum outputs. Planning as modelling means that the modelling system probabilistically predicts the future inputs and outputs of the AI system. The modelling system attempts to predict both external events in the real world and its own behaviour, with no distinction between them. The AI system's *predictions* of its own future behaviour are equivalent to *planning* of its own future behaviour and can be used to constrain a search for optimum behaviour to such a degree that almost all of the work involved in such a search is already done by the application of the constraints. Planning therefore occurs almost completely within the modelling system and is reduced to a trivial special case of modelling. Planning is prediction.

One way of implementing this is as a tree search that experiments with possible sequences of outputs, much as a chess algorithm experiments with possible sequences of moves [11,12], to determine those outputs that are best to make now, with more attention

being given to those paths through the search tree which represent more likely behaviour by the AI system, so that such paths are explored to a greater depth.

Rather than meeting the specification of the modelling system described in previous articles [3,4,5,6], it is merely required that the modelling system fulfil very general criteria. The modelling system must make probabilistic predictions of the future inputs and outputs of the AI system, based on its previous inputs and outputs and the modelling system must not distinguish between inputs and outputs: outputs are treated as inputs, so that the modelling system is observing the AI system's behaviour as just another aspect of reality.

These criteria could be met by various modelling systems. In fact, they would be met by any modelling system that we should take seriously as part of a general solution to AI.

The modelling system needs to be able to prioritize its use of limited computing resources. This is dealt with by prioritization control outputs. Prioritization control outputs are special outputs that, instead of influencing events in the outside world, are used to control prioritization in the modelling system. The use of outputs to control prioritization means that prioritization is learned in the same way that conventional behaviour is learned. One way of viewing this is the *AI as a boundary system* view [7] in which, as far as outputs are concerned, the modelling system is just another part of external reality manipulated by the AI system to improve its situation. Such prioritization is an important part of what the model is.

References

- [1] Web Reference: Almond, P. (2006). *How AI Would Work*. Retrieved 4 September 2006 from <http://www.paul-almond.com/HowAIWouldWork.pdf>.
- [2] Web Reference: Almond, P. (2006). *Occam's Razor Part 9: Representation and Planning of Actions in Artificial Intelligence*. Retrieved 29 July 2006 from <http://www.paul-almond.com/OccamsRazorPart09.pdf>.
- [3] Web Reference: Almond, P. (2006). *Occam's Razor Part 6: Partial Models as "Envelopes"*. Retrieved 1 March 2006 from <http://www.paul-almond.com/OccamsRazorPart06.htm>.
- [4] Web Reference: Almond, P. (2006). *Occam's Razor Part 7: Hierarchy and Ontology*. Retrieved 30 April 2006 from <http://www.paul-almond.com/OccamsRazorPart07.htm>.
- [5] Web Reference: Almond, P. (2006). *Occam's Razor Part 8: Modelling in Artificial Intelligence*. Retrieved 9 June 2006 from <http://www.paul-almond.com/OccamsRazorPart08.pdf>.

- [6] Web Reference: Almond, P. (2006). Downward Transfer of Probabilities in AI. Retrieved 15 October 2006 from <http://www.paul-almond.com/DownwardTransferOfProbabilitiesInAI.pdf>.
- [7] Web Reference: Almond, P. (2006). AI as a Boundary System. Retrieved 17 September 2006 from <http://www.paul-almond.com/AIAsABoundarySystem.pdf>.
- [8] Hawkins, J., Blakeslee, S. (2004). *On Intelligence*. New York: Henry Holt.
- [9] Web Reference: George, D., Hawkins, J. (?). *Belief Propagation and Wiring Length Optimization as Organizing Principles for Cortical Microcircuits*. Retrieved 24 April 2006 from <http://www.stanford.edu/~dil/invariance/Download/CorticalCircuits.pdf>.
- [10] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 2, pp7-37.
- [11] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 3, pp38-52.
- [12] Heinz, E, A. (2000). *Scalable Search in Computer Chess: Algorithmic Enhancements and Experiments at High Search Depths*. Vieweg Verlag. Chapter 0, pp11-18.
- [13] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 4, pp58-59.
- [14] Web Reference: Almond, P. (2003). *Indirect Mind Uploading: Using AI to Avoid Staying Dead*. Retrieved 9 August 2003 from <http://www.paul-almond.com/IndirectMindUploading.htm>.