

Occam's Razor Part 8: Modelling in Artificial Intelligence

By Paul Almond, 6 June 2006

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

© Copyright Paul Almond, 2006. All Rights Reserved.

Occam's Razor Part 8: Modelling in Artificial Intelligence

By Paul Almond, 9 June 2006

Previous Articles in this Series

This article is part of a series. These are the previous articles:

Occam's Razor Part 1: What Is Occam's Razor? <http://www.paul-almond.com/OccamsRazorPart01.htm>. [1]

Occam's Razor Part 2: Principles of Language <http://www.paul-almond.com/OccamsRazorPart02.htm>. [2]

Occam's Razor Part 3: Assumptions About Reality <http://www.paul-almond.com/OccamsRazorPart03.htm>. [3]

Occam's Razor Part 4: An Overview of How Occam's Razor Works <http://www.paul-almond.com/OccamsRazorPart04.htm>. [4]

Occam's Razor Part 5: How Mapping Can Work <http://www.paul-almond.com/OccamsRazorPart05.htm>. [5]

Occam's Razor Part 6: Partial Models as Envelopes <http://www.paul-almond.com/OccamsRazorPart06.htm>. [6]

Occam's Razor Part 7: Hierarchy and Ontology <http://www.paul-almond.com/OccamsRazorPart07.htm>. [7]

In particular, the last two of these, parts 6 and 7, may be helpful in understanding this article.

The following article, although not part of this series, is closely related to it:

What is a Low Level Language? <http://www.paul-almond.com/WhatIsALowLevelLanguage.htm>. [8]

Introduction

Previous articles have proposed an ontological system based on partial models extracting meaning from data. A partial model is an algorithm. Previous articles suggested that such an ontological system can be used in artificial intelligence to allow a computer system to extract meaning from its environment, but no detailed description has been given of how this will work. This article will do that: it will describe how the ontology described so far can be applied in an algorithm that extracts meaning from reality. Such an algorithm would be a formalized version of Occam's razor.

This article is less philosophical than previous ones: here, I am purely concerned with making artificially intelligent machines.

This article will not come close to solving all the problems of artificial intelligence. It will make a proposal for solving one particular one: how a computer can use sensory inputs obtained from reality to assemble a worldview - a model of how reality works. We will be exploring this in some detail. Unlike previous articles we won't just be making general references to this sort of thing. I will be describing this in enough detail to allow it to be tested in software.

The Broad Idea

We start with sensory inputs from reality. We regard these as being one or more matrices of bits. We have a number of partial models. Partial models are meaning extraction algorithms. Each partial model algorithm looks for a particular meaning in the matrix or matrices of data. It can be applied for different locations in the matrix by changing the index of the partial model. When a partial model algorithm finds the relevant meaning it indicates this with its output. This is an “object detection”. When a partial model determines that there is no such meaning to be found it gives an output indicating failure to detect the object.

A particular partial model will indicate object detection success for some values of its index and failure of object detection for others. This pattern of successful and failed object detection can form another matrix - a layer of meaning. We can regard this as an extra layer of reality, generated by the partial model algorithm. Instead of having to consider this in an abstract way, it might be helpful to consider this as being “overlaid” on the original input data. This is a simplified, visual model that does not completely describe the situation, but it may help. Rather than consider this new layer of meaning, or extra, synthetic layer of reality, abstractly we can imagine the pattern of successful and failed object detection as being marked with an overhead projector pen on a transparent piece of plastic and overlaid on top of the original matrix. When we look at this now we see the original data and a pattern of “1”s and “0”s showing where a particular pattern has been detected in that data. We can do this for a number of partial models, stacking many pieces of transparent plastic, each showing a different meaning. This could give us an idea of the distribution of the occurrence of patterns in that data. (See Figure 1)

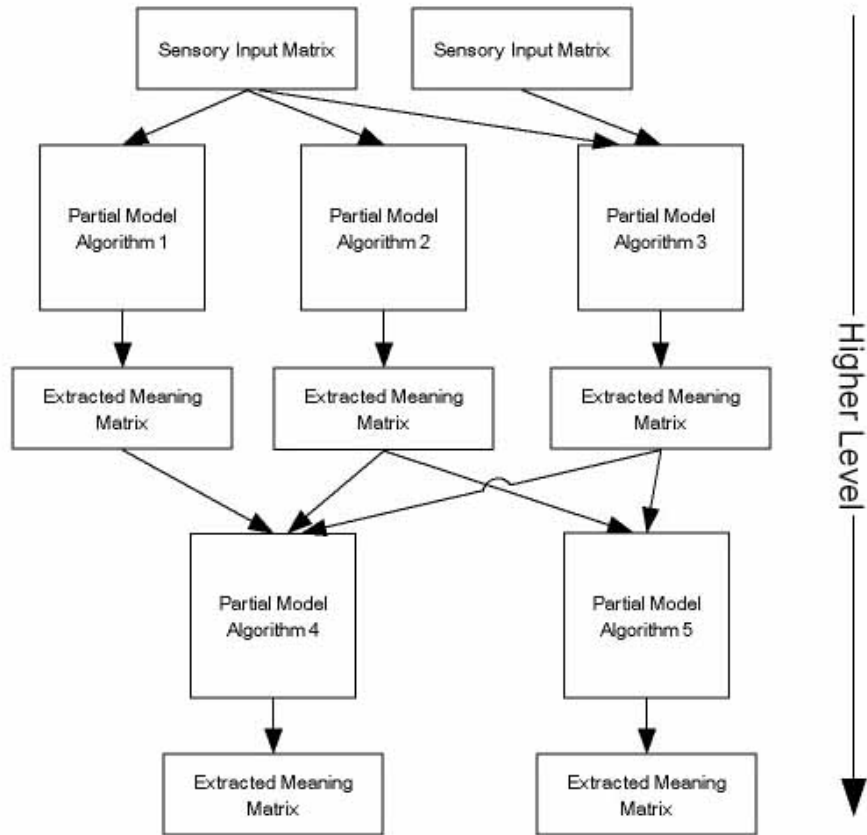


Figure 1: Hierarchical Extraction of Meaning by Partial Model Algorithms

We are not limited to extracting meaning from the original data: we can also extract meaning from the created reality that we have made. We could make a partial model that finds patterns within one or more of these “transparent sheets”. The new layer of reality would consist of a pattern of “1”s and “0”s showing the occurrence of this higher level pattern and we could overlay that transparent sheet on the previous pile. We can do this for a number of partial models which extract meaning from previously extracted meaning and continue doing this. As our stack of transparent plastic sheets gets higher some of them would show the distribution of patterns based on extraction of meaning many times removed from the previous data, because it is based on meanings extracted from lower sheets, which were extracted from still lower sheets and so on.

Any pattern extracted in such a way is regarded as an *object*. An object is simply the occurrence of a “1” on one of these layers. This is a rather abstract use of the word “object”. An object does not have to be a solid “thing”, as we think of objects as solid things in everyday life, nor does it have to be spatial: objects could be detected by partial model algorithms in all kinds of matrices.

If a matrix of sensory inputs from which we are extracting meaning represents data acquired over a period of time then the patterns that we extract will not just relate to *things*. They will relate to spatial objects *and temporal* events in reality. An object could

be a pattern existing in space and over a period of time. From this point of view, cats, computers, wallpaper patterns, the kicking of a ball and World War II are all objects.

In previous articles I discussed a number of different partial model algorithms – all essentially the same thing used in different ways. The focus in this article is on *object-like* partial models. Such a partial model algorithm can be described as follows:

$$R_i = M_i(P_i, \mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_n, \mathbf{i})$$

where:

M_i is a meaning extraction algorithm – the partial model algorithm itself.

R_i is the result returned by the partial model algorithm for some index i . For the partial model algorithms that we are considering, this result will be “1” for successful object detection or “0” for failed object detection.

P_i is a parameter which does not serve any purpose at the moment, so I suggest you just ignore it. I included it in case more information was needed in the system later.

\mathbf{i} is a vector describing a location in a matrix or matrices.

$\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_n$ are the matrices of previously extracted meaning on which the partial model operates.

The result, R_i , generated for different values of the index i implies a new matrix and other, higher level partial model algorithms can extract meaning from it.

Comparison with Bayesian Networks

This article will certainly make use of Bayesian ideas about probability. Superficially, what is proposed here may seem like a Bayesian network, but there is an important difference.

In a Bayesian network nodes are variables and the “edges” (connections between variables) are probabilistic in nature. A Bayesian network describes probabilistic relationships.

In the ontology in this article the nearest things to nodes are the bits in the matrices and these represent extracted meanings in a hierarchy. The nearest thing to connections between nodes is the dependence of an extracted meaning bit on the bits in lower levels of the hierarchy that are read by the partial model algorithm that did the extraction. This dependence is not probabilistic at all, unlike that in a Bayesian network. This is because there is no probabilistic nature in a partial model algorithm: it defines the conditions required by lower level data for a high level meaning to exist in a non-probabilistic way.

This does not mean that probability is unimportant in this ontology. The difference is that it will not be the *relationships* that are probabilistic but the extracted meanings themselves, due to uncertainty in the sensory inputs from which the meanings are extracted. The whole structure of meaning extraction which produces these meanings will

be completely deterministic and will generate probabilistic results when applied to uncertain observations.

All meanings are, in the beginning, extracted from sensory inputs. If all the sensory inputs from which we want to extract any meaning were known, including those that could be made in the future, then there would not be any uncertainty anywhere in the hierarchy, which would not be the case with a Bayesian network. We cannot have perfect observation of reality, however, and when there are omissions in observation uncertainty appears in the extracted meanings.

I think that this is doing things the right way round. The partial models that extract meanings are *definitions* of patterns and definitions should not be probabilistic.

As an example, let us suppose that we have a hierarchy of partial model algorithms which extract the meaning of “a cat” from sensory inputs. These partial model algorithms relate observations of reality to the concept of a cat. It is these partial model algorithms that describe what a cat *is*. To suggest that our concept of what a cat is should be probabilistic makes no sense - we *know* what one is – and so the relationships in a hierarchy of meaning extraction should not be probabilistic. The uncertainty appears when we try to apply this concept of what a cat is to incomplete observations.

In many situations we may be unsure about whether a cat is there. We may hear something that could be a cat. We may see a small part of a cat through gaps in vegetation. Even if we can see a cat in front of us we may wonder if it could somehow be a “fake” cat – maybe with robotic parts inside. In these situations none of the uncertainty comes from how the concept of a cat relates (down the hierarchy) to observations. It comes the incompleteness of our observations and, provided that the concept of what a cat is – the set of meaning extraction rules for it – is well-formed, we should always be able to resolve the issue with certainty, if we are allowed to obtain unlimited information from reality. We are not omniscient, however, so uncertainty is produced when we try to relate our deterministic concept of what a cat is to our incomplete knowledge of reality.

The ontology discussed here is not a Bayesian network.

Inference

The Idea of Inference

An important process in the modelling system is *inference*. Inference is the extraction of meaning from matrices of bits when some of the bits read by the partial model algorithm have unknown values.

As it has been described so far, a partial model reads bits from matrices and returns a meaning. The meanings returned in this way for application of the partial model with many different indices can be used to construct a further matrix. If all of the bits in the matrices are known then this is a straightforward process.

Complications arise when we do not know all of the bits in the matrix. A partial model, running to extract meaning from a matrix for some index i in such a situation, may attempt to read bits from a matrix which have unknown values. The partial model algorithm “expects” to know the values of these bits to work and return a result of success or failure in object detection. In such a situation there is uncertainty about the partial model result and some sort of statistical process is needed. This is the process that I will call *inference*. So far, I have not proposed any method of inference: the partial models have needed knowledge of all the bit values that they read. This article will deal with this issue.

Inference can be thought of as similar to what we do when we see part of a known object or pattern and realize that the whole object or pattern is there. Inference is important because any artificially intelligent computer system would only be able to observe a small part of reality: if it could not know about unobserved parts of reality it would know almost nothing. Inference is also important with regard to time and prediction.

Inference and Time

If the observations made by an artificially intelligent system are stored in one or more matrices then the matrices will contain bits for observations made in the past. They will not contain any of the observations made in the future for an obvious reason: they have not been made yet. All of the parts of the matrices dealing with observation in the future therefore contain bits with unknown values. Attempting to extract meaning from these parts of the matrices is therefore equivalent to predicting the future. Prediction of the future, the main purpose of a modelling system, is just a special case of inference.

How can we extract meaning from empty parts of the matrix?

It may be easy to accept that an inference process (which I have not yet described) can extract meaning in a statistical way when *some* observations are available for a partial model algorithm to read, but an artificially intelligent system has *no* observations made in the future. It may be harder to see how meaning can be extracted from the future when it is totally empty of any observations. What would any inference process have on which to base such meaning extraction?

Inference in such a situation would work by extracting meaning in the near future and gradually extending objects into the future. Recent observations can be used to extract higher level meanings for objects corresponding to spatial and temporal events that have been happening for some time and will *continue* to happen for some time in the future. Higher level partial models can be used to further extract meanings from these objects and these new meanings will correspond to events that extend still further into the future.

An analogy for future prediction is the problem of working out how the pattern of a piece of wallpaper continues beyond the end of the sheet. The fact that there are parts of the wallpaper that you cannot see is not an obstacle. Where the edges are you will see parts

of patterns – cut off by the edges – and these will allow you to deduce the existence of complete patterns off the edge of the wallpaper. These will be parts of still higher level patterns, allowing you to extend the patterns into parts of the wallpaper that you have not seen. Predicting the future is essentially like extending a “wallpaper pattern” of space-time events.

Downward transfer of probabilities, which will be discussed briefly later in this article and developed in further articles, will also play a part in prediction of the future. It will allow some of the patterns generated at higher levels as predictions of the future to be used to make predictions at a lower level.

The Importance of Probability

Probability will be a big feature of any viable artificial intelligence modelling system because:

- Probability is an intrinsic part of human modelling. We have varying degrees of belief in claims about reality.
- The process of inference described above is statistical, so the results of inference should be expressed probabilistically.

Beyond this, I have other reasons for thinking that probability is an important concept. A modelling system is only part of what an artificially intelligent computer system needs. It also needs some way of determining how to act. For reasons that I will explain in a later article, I think that the “correct” methodology for planning can only be coherently expressed within the context of a probabilistically expressed ontology. This means that a supposedly intelligent computer system that lacks a probabilistically expressed ontology will be deeply flawed as the way that it plans its actions would have to work in a totally different way to that of a system with the correctly expressed, probabilistic ontology. This problem would not be rectified by shallowly introducing a probabilistically expressed worldview into such a system: it is so fundamental to how the system should work that it needs doing in the design stage.

I also think that the problem commonly known as the *carpet texture* problem has a solution in a hierarchical system like this that relies on the capability to plan actions. As I think that the planning of actions needs a probabilistically expressed worldview then a solution of the carpet texture problem would also depend on this.

For these reasons, probability is an integral part of the method proposed in this article. To summarize this:

The language of both worldviews *and* planning of actions is the language of hierarchies of *probabilities*.

Probabilistic Representation of Data

A matrix of observations contains bits – “1”s and “0”s. When we store a representation of any such matrix, however, we will not be storing “1”s and “0”s. This is because we will not have certainty about what most of the bits should be.

In the “bottom level” of the hierarchy – the matrices of bits corresponding directly to sensory inputs - there is certainty about some of the bits, although other bits will have unknown values as they correspond to future sensory inputs that the system has not received *yet*. We could store this data in a computer by having three possible values for each bit: “0”, “1” or “?” (“unknown”), so that the matrices contain “0”s and “1”s for sensory inputs that have occurred and “?”s for sensory inputs that have not yet occurred.

At higher levels in the hierarchy there will be more uncertainty. The unknown bits will result in object detection or failure by partial models being *inferred*. For any of these bits the computer will only have a *probability* that it has a given value. When bits about which there is such uncertainty are read by a partial model still further uncertainty will be generated at higher levels. We cannot rely on the previous idea of bits being known or unknown. At higher levels in the hierarchy there may be some uncertainty about the values of almost all the bits, but there will be varying degrees of uncertainty.

The solution is to represent the bit values as probabilities rather than “1”s or “0”s. A value of 0 would indicate a probability of 0 that the bit was a “1” (that is to say, certainty that the bit was a “0”) and a value of 1 would indicate a probability of 1 that the bit was a “1” (that is to say, certainty that the bit was a “1”). Any value between 0 and 1 can be stored to represent what we know about a bit. Where there is complete uncertainty about a bit a value of 0.5 would be stored for it, indicating that it is equally likely to be “0” or “1”.

On the bottom level of the hierarchy, for the matrices of sensory inputs, it may seem that we do not need such a representation of data. We do not have the varying degrees of probability that are generated by inference at higher levels and it may appear that for this level we can just store values of “0”, “1” or “?” (“unknown”) for the bits. The process of inference, however needs to be standardised. It will not just involve generation of probabilities for bits at higher levels but will also involve using the information stored about bits at lower levels. For this reason, the way in which information is stored in the matrices should be the same for all of them, including the matrices of sensory inputs. Converting the “0”, “1” and “?” (“unknown”) values to probabilities is simple. If a bit is known to have a value of “0” then we store a probability of 0 for it and if it is known to have a value of 1 then we store a probability of 1 for it. If the value of a bit is “?” (“unknown”) then we store a probably value of 0.5. The use of such a system means that all the matrices of sensory inputs and extracted meanings contain probabilities for bit values rather than bits.

It should be noted that this does not imply any change in the ontology described previously. We are simply storing probabilities to describe limitations in our knowledge.

There is no change in how partial model algorithms work. They still read bits from the matrices – not probabilities - and they output bits – not probabilities - to indicate success or failure in object detection for a particular index. This means that a partial model could never be run directly on the probabilistic versions of the matrices stored in the computer system. The problem will be dealt with by the process of inference and we can now restate the process of inference in a better way with the *probabilistic description of inference*:

The Probabilistic Description of Inference

A partial model algorithm with some index operates on a matrix or matrices of bits and extracts a meaning corresponding to successful object detection or failure in object detection.

The matrices representing reality in a computer will be *probabilistic matrices* and will store probabilities of bit values instead of bits.

Inference is the process of determining the probability that a particular partial model algorithm, used with a particular index, would output a successful object detection, when the matrices from which it is extracting its meaning are probabilistic.

As the results of inference are probabilities they can be used directly to assign probabilities to other bits at higher levels in the hierarchy.

From this, it follows that inference cannot rely on just running a partial model algorithm. A partial model algorithm can never even be run on the probabilistic matrices and is now merely a conceptual algorithm. That does not mean that there is any vagueness about a partial model. It is a properly expressed algorithm that would really work and generate a decision *if* we knew the actual values of all the bits in the matrices, instead of just probabilities. The fact that we rarely, if ever, have this information means that it hardly ever or never gets run. Instead, the process of inference uses this conceptual algorithm to determine the probability that it would give an object detection result if we did have this information and ran the partial model algorithm

There is now no real distinction between situations in which we would run the partial algorithm to generate a result and situations in which inference is needed to deal with probabilities. There are no “unknown” or “known” bits now: there are just probabilities. There may be situations in which we know all the bit values needed by a partial model, but in most real world situations these will be so rare that there is no point in having any “conventional” meaning extraction as well as inference. Inference can be used for everything. If a situation were encountered in which all the bits that could be read by a partial model had known values (that is to say, probabilities of 0 or 1) then the result of inference would be a probability of 0 or 1 and this would be equivalent to just running the partial model anyway. Just running the partial model can be considered to be a special

case of inference in situations when the bit values are known and we do not need to deal with it specifically: a viable inference process will deal with it automatically.

Comparison with the Neural Hierarchy of Jeff Hawkins

There is some similarity between the hierarchical ontology discussed in this article and the neurological model proposed by Jeff Hawkins [9,10]. It uses the same principles of hierarchical extraction of patterns and what Hawkins calls *concept invariance*.

One difference is that the processes discussed here are not specific to neural systems. That in itself is not too significant: they could be mapped into a system resembling a neural one or any number of other systems and Hawkins has probably considered his own method in a general way, outside the scope of neurology, simply selecting neurology as a preferred method of implementation.

The real advantage of the process in this article is how it deals with probability. I do not think that there is adequate provision in the Hawkins method for this. As I have already stated, I think this is an important matter. Having probability in the ontology is an important part of getting it right, not just because information about reality will be limited and will need to be described in probabilistic terms, but because the ways in which the system plans actions is defined will need to be probabilistic in nature. A solution to the carpet texture problem will also need a probabilistic ontology.

I think the ontology and process discussed here provide a better consideration of probability, and so provide a better way of doing things, both in terms of more accurately describing what is known about reality and in being extendable to make a working artificially intelligent system.

I will now explain how the process of inference works. Inference uses the idea of a *decision tree* and various concepts relating to probability.

The Decision Tree

The Concept of the Decision Tree

A decision tree is a conceptual structure mapping every possible route through a partial model algorithm running with a particular index value.

When a partial model algorithm is run with a particular index value there is one of three possible outcomes:

1. The partial model announces successful detection of the object and stops.
2. The partial model announces failure to detect the object and stops.
3. The partial model does not stop.

We do not need to worry about this third possibility right now. As the partial model algorithm runs it can read bits from the matrix of observations (or from one or more matrices of previously extracted meaning, if it is at a higher level in the hierarchy) Apart from when it makes its outputs, the bits that the algorithm reads are its only interaction with anything beyond itself, so they completely determine the state that the algorithm is in and the final output that it gives.

This means that each time a bit is read it is a decision point in the future of the algorithm run. If a “0” is read the algorithm may go one way and if a “1” is read then it may go another way.

We can describe all the possible progressions that could occur through the algorithm by describing every possible sequence of bits that the algorithm could obtain by reading the matrix. For example, two different runs for a partial model algorithm, with the same index, could start as follows:

- The algorithm starts and reads a “1” from somewhere in the matrices. It then reads a “1” from another location in the matrices, followed by a “0” from somewhere else, and so on.
- The algorithm starts and reads a “0” from somewhere in the matrices. It then reads a “0” from another location in the matrices, followed by a “1” from somewhere else and so on.

I have not said from where in the matrices the partial model algorithm is getting these bits. That is because we do not need to know. The algorithm gets them from wherever its instructions tell it to get them. All that concerns us is the index given to the partial model algorithm and the output – success or failure in object detection - that it returns. For the two example runs described above, the first bit read by the partial model must be in the same place in both instances because the state of the algorithm run is the same in each instance: it is the same partial model started with the same index. After this first bit has been read then the two algorithm runs are in different states – for example, some decision command in the algorithm may tell it to go one way or another based on the value obtained for that first bit or some other information derived from it – and this means that the second and subsequent bits need not be read from the same place.

We can express the different states that a partial model algorithm can enter during a run with a *decision tree*. The decision tree starts at a single node representing the algorithm starting to run and reading its first value from the matrices. Two branches split off downwards from this node – one for the algorithm reading a “0” and the other for it reading a “1”. Each of these branches leads to another node on the next level of the tree representing the second time that the algorithm reads a bit from the matrices and the bits may now be read from different places. Each of these nodes has two branches going down to the next level - one for obtaining a “0” the second time a bit is read and one for obtaining a “1” and so on.

This is how the decision tree is constructed. It grows downwards as the algorithm reads bits. Each time the algorithm reads a bit from the matrices there is a node with two branches leading down from it for obtaining “0” or “1”.

A branch of the decision tree terminates – ends in a node without any branches leading down from it – if the algorithm gives an output and stops. When a branch ends in this way, therefore, it ends with the partial model algorithm’s output indicating successful object detection (“1”) or failure to detect the object (“0”).

The decision tree must be consistent. If the partial model algorithm reads a bit from the same place in the matrices twice then it must obtain the same value on each occasion. This means that where there is a node corresponding to a bit which has already been read (higher up at a previous node in the decision tree) then that bit will already have a known value, which must now be used again. Such nodes will therefore only have a single branch, corresponding to the value that was obtained previously for the bit. A node with only one branch is redundant and can be removed from the decision tree.

Some branches of the decision tree could go on for a long time, or even forever. In practice there will be a limit on how deep we can make the decision tree and we would have to terminate branches which have gone too far at some point, as if the algorithm had actually terminated. For such branches we will have a “null result” node instead of a node at which there is success or failure in object detection. (See Figure 2).

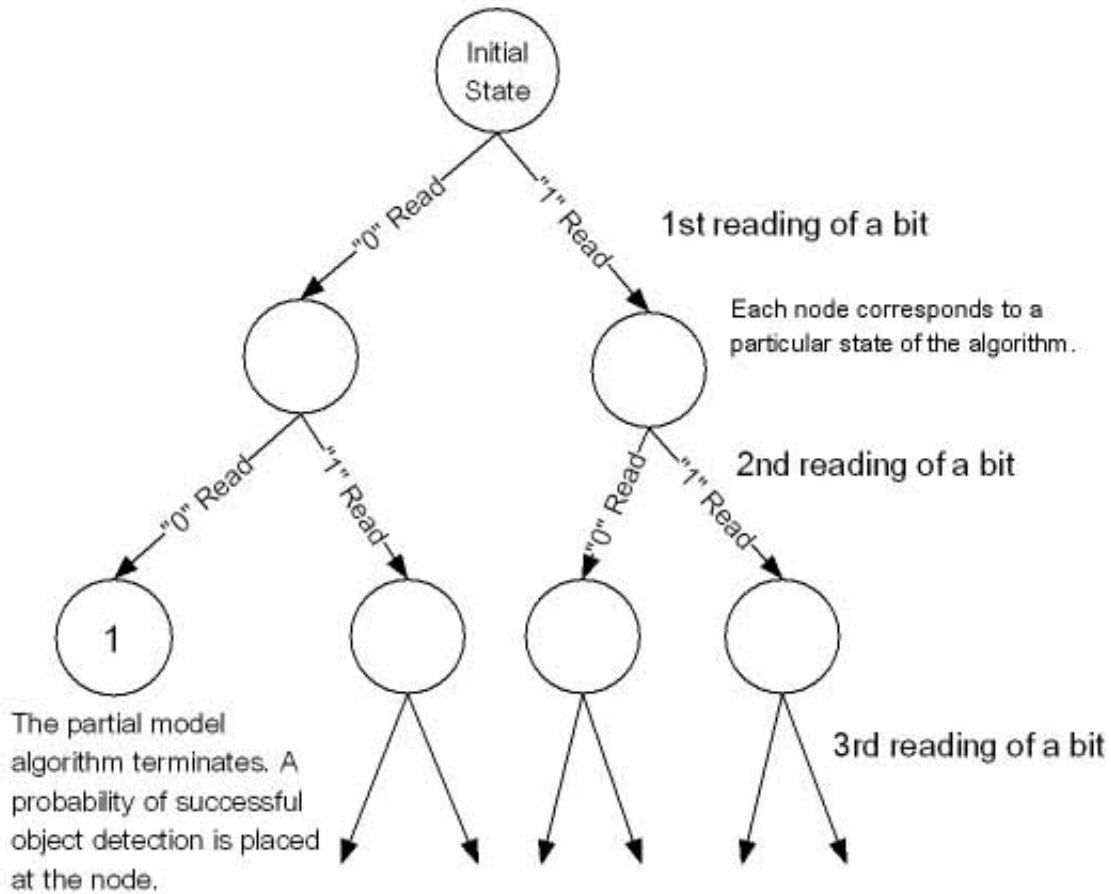


Figure 2: A Partial Model Algorithm Decision Tree

Constructing a Decision Tree

The process of construction need only be done once for each partial model algorithm: the same decision tree can be reused any number of times and for different indices.

To construct a decision tree we start at the top node of the tree, at the start of the partial model algorithm and before the algorithm has read any bits from the matrices, and execute the following recursive process at this node:

We step through the partial model algorithm from whatever point execution of the algorithm had previously reached (that is to say, we simulate running it) until it attempts to read a bit from one of the matrices. We first assume that the bit value obtained was “0”, create a “0” branch from the current node, with a new node on the end of it, and execute this process recursively to extend the decision tree from this new node. We next assume that the bit value obtained was “1” *instead*, create a “1” branch from the current node, with a new node on the end of it, and execute this process recursively to extend the decision tree from this new node.

There are some exceptions to this, which follow from what I have previously said about decision trees:

- If, during simulation of the running of the partial model algorithm, it outputs a result (“1” for successful object detection or “0” for failure in object detection) then we do not extend any further branches from the current node. We simply make the current node act as a termination of the decision tree and place a probability at this node. This is the probability of the decision tree outputting a successful object detection. If a successful object detection (“1”) were output we use a probability value of 1 and if a failure in object detection (“0”) were output we use a probability value of 0.
- I have previously said that the decision tree must be consistent. If a bit is read from a particular position in a matrix and a bit is read from the same position again then the same value must be obtained every time. This means that each time a bit is read from a matrix we should check to see if it has already been read higher up in the decision tree. If it has been read then the previous value should be assumed and after that the reading of this bit should be ignored. No new node needs to be created as such a node would only have one branch and would be redundant.
- I have previously said that some branches of the decision tree could go on for a long time, or even forever. Limits on computing power and time impose limits on how deep the decision tree can be allowed to go. When some maximum depth is reached at a node then no further branches are extended from it: instead it is assigned a “null result” label instead of a probability.

Cleaning Up a Decision Tree

This is really the part of the decision tree construction process, but I am describing it separately so that what I have previously said about construction will be easier to understand. Cleaning up of a decision tree involves dealing with “null result” nodes and removing redundant branches from the decision tree to increase the efficiency with which it can be used. The cleaning up could be performed as a separate process, but it could also be embedded into the recursive construction process described above.

When the recursive process has executed for a node it should send details of the outcome of its execution (the value that was placed at the next node down) back up to the level of recursion that “called” it. This means that any node from which branches are extended downwards receives two values back. If the following special cases occur they are dealt with as described:

- If both of the branches extended down by the recursion process from the current node (for reading “0” or “1” as the current value for a matrix bit) result in nodes at which the decision tree terminates and the probability values at both of these nodes are the same (that is to say, both 0, both 1 or both “null result”) then the decision tree is made to terminate at the current node and the branches and nodes

below it are erased. The current node is assigned whatever value was on both of the lower nodes where the tree terminated.

- If one of the branches extended down by the recursion process from the current node (for reading “0” or “1” as the current value for a matrix bit) results in a node at which the decision tree terminates with a “null result” value then the current node is removed and the branch which extended from it and did not lead to the “null result” node is connected to the node above – equivalent to assuming that the path down the “null result” branch can never occur and then simplifying the decision tree accordingly.

If, after completion of the decision tree, the first node is found to have a “null result” label then it is assigned a probability value of 0.5 instead. This would happen for a partial model which was never observed to return a result, either because it just runs forever or because any results that it returns occur at such high depths in the tree that they are inaccessible with available computing power. A partial model algorithm which behaved like this would be useless, raising the question of why we would bother cleaning it up. Ensuring that all “null result” nodes are removed, however, helps to prevent possible program crashes when systems are using many partial models.

My approach to dealing with “null result” nodes may be controversial. It has the advantage of just using the information that we do have for probability generation. A disadvantage is that, in some situations, it could suggest less uncertainty than should be assumed. As an example, if the cleaning up process took most of a decision tree away, leaving just a node with branches leading down to two other nodes (this will not happen often in practical situations), one of these nodes having a probability of 1 and the other node being “null result” then the “null result” node would be discarded and both of these nodes would be removed, with a probability of 1 being placed at the top node. This would suggest that there is certainty that this partial model would always indicate successful object detection, yet the fact that we do not really know what would happen if the algorithm state developed down the “null result” path indicates that a failure in object detection could happen. I do not think that this would be much of an issue as “null result” nodes should generally be taken out of the decision tree much lower down, where whatever we do just becomes part of the statistical noise.

Another approach to “null result” nodes might be to assign them all the same probability – an average of all the other probabilities at terminating nodes. This could also tend to underestimate uncertainty in some situations.

A further approach could be to assign all “null result” nodes a probability of 0.5 but this could mean assuming more uncertainty than is needed. To give a contrived example: if we had 999,999 nodes all involving termination with a probability of 1 and one “null result” node we may think that this node had a probability greater than 0.5 of having a “1” output so such a method could assume more uncertainty than we need to assume in some situations. This would, however, be safer than assuming too much certainty.

More consideration should be given to the “null result” situation in future, but the method that I have chosen is adequate for now.

Collapsing a Decision Tree

The collapsing of a decision tree is the process of using it to obtain a single probability value, corresponding to the probability of successful object detection by a partial model, given a particular index value and one or more matrices from which it extracts its meaning.

Decision tree collapse is not inference. It is a process used in inference. The probability values generated by decision tree collapse are intermediate results and are *naive* – I will explain what this means shortly – and the process of inference requires the decision tree to be collapsed more than once.

Decision tree collapse uses a recursive algorithm which traces through all the paths of the decision tree in the same way in which the Shannon look-ahead chess algorithm [11,12] traces through possible sequences of moves. We start at the top of the decision tree, at the stage before the algorithm has attempted to read any bits from the matrices, and execute the following recursive process:

When we are at a node with two branches – one branch corresponding to reading a value of “0” and the other value corresponding to reading a value of “1” we first assume that we read “0” – meaning we follow the “0” branch down to the next node – and then we execute this process recursively again. Calling this process recursively returns a probability value (that is to say, it passes it back up the decision tree) and we store this value which we will call *the returned “0” probability*. Then we assume that we read “1” *instead* and go down the “1” branch to the next node and execute this process recursively. Calling this process recursively returns a probability value (that is to say, it passes it back up the decision tree) and we store this value which we will call *the returned “1” probability*.

When we have the returned “0” probability and the returned “1” probability we know the probabilities of successful object detection if the partial model algorithm reads a “0” or “1” from the matrices at this point. We now need to determine the probability of the algorithm reading a “1” from the matrix at this point and this value is simply the probability stored in the place in the appropriate matrix at which the partial model algorithm is attempting to read a bit. We simply look this value up and we will call it *the probability of reading “1”*.

When this process has been executed recursively for both the “0” and “1” cases we have two stored probabilities that were returned by the recursive calls for the “0” and “1” branches – the returned “0” probability and the returned “1” probability. We also have the probability of a “1” being read from the matrices at this point and this also tells us what the probability of reading a “0” from the matrices at this point is: it is 1-the probability of reading “1”. This allows to calculate a single resultant probability as follows:

Resultant Probability = (probability of reading “1” x returned “1” probability) + (probability of reading “0” x returned “0” probability)

This probability value is then passed back up as the returned probability from this instance of execution of the recursive process.

In this way, the recursive process traverses all the branches of the decision tree and returns a single probability value.

There is one exception to this:

If the recursive process is executed for a node at which the decision tree terminates (that is to say, there are no branches leading downwards from this node) then there will already be a probability value stored at this node. In such a case, we simply return this probability value from the recursive process.

It should be noted that:

- We do not need to be concerned about consistency when a bit is read from the same position in a matrix twice. Such situations have already been dealt with during the construction of the decision tree and no such nodes will exist after cleaning up.
- The partial model algorithm is not needed to collapse its decision tree: it is only needed during the construction of the decision tree, which then contains all the required information about it.

Naive Probabilities

The process involves obtaining two sorts of probabilities relating to the object: *naive object probabilities* and *naive algorithm probabilities*. Many people would consider these abstractions rather than real probabilities. For now, we can assume that they have little practical meaning in the real world and are computed as intermediate results to allow us to obtain the final, *real* probability that the object exists – though we will return to this later in the article.

Naive Object Probabilities

These are probability values relating to the existence of the object that do not take into account Occam’s razor. That is to say, they do not consider the length of the partial model algorithm needed to confirm existence of the object or the size of the algorithm needed to describe or construct it. A naive object probability can also be computed when we have partial information about a pattern that may or may not be the object and it will be affected by any possession of such information.

Naive Algorithm Probabilities

These are probabilities relating to the existence of the object that do not take into account anything considered when generating a naive object probability as described above. If naive object probabilities are naive because they lack consideration of Occam's razor then naive algorithm probabilities are naive because that is *all* that they consider. A naive algorithm probability for an object's existence is generated purely from the length of the partial model algorithm needed to confirm its existence, or from the length of an algorithm needed to describe or construct the object.

Why are these types of probabilities described as naive?

They are calculated without looking at the overall picture. They are the sorts of probabilities that would be computed by people who are competent, yet have narrow knowledge about what contributes to the expectation of an object's existence.

An example may help to explain this:

Suppose we saw this sequence of six binary digits in a much longer sequence:

010101... and we have not yet seen what comes next.

We want to know if that small sequence was part of this longer sequence of twenty bits:

01010101010101010101

This twenty bit sequence is the object in which we are interested.

Let us imagine we have two mathematicians who are naive about probability, meaning that their skills at dealing with it are limited to a particular domain, yet each is entirely competent within that domain.

We will call the first mathematician a *naive objectivist*. This mathematician only considers the object itself. This causes any probability values that he generates to be naive object probabilities. He tells us that the probability of the object occurring at any point in the sequence is the combined probability of each individual bit occurring, assuming that each bit has a 0.5 probability of being the correct value. The probability of the entire sequence occurring is therefore 0.5^{20} (about 9×10^{-7}). When we know that we already have part of the sequence the existence of the object will not be quite as unlikely. We already have the first six bits so all that is now required is for the remaining fourteen bits to be correct, and the probability of this is 0.5^{14} (about 6×10^{-5}).

If all the bits in the sequence were random then the naive objectivist would be correct about all this. If, however, Occam's razor is correct then the sequence, or whatever arrangement of bits is being observed, is not random: there is a bias in favour of some objects based on the lengths of the algorithms needed to describe them. In such a

situation (and the universe appears to be one) this probability estimate may be very naive: it may suggest that an object is unlikely, because it needs a large number of bits needed to make it, when the object may actually be likely because it is described by a short algorithm. Occam's razor is totally lost on the naive objectivist. To us, seeing half of a cat, the other half being obscured, generally suggests to us that we are probably seeing part of an entire cat that exists. The naive objectivist would not see things like this. He would say that, for a cat to exist, lots of atoms have to be in the right places and this is very unlikely. He would accept, however, that the existence of a cat is more likely given that he can see half of a cat: he already knows that about half of the atoms are in the right places. For the rest of the cat to be there, however, he would say that a lot of other atoms (about half the number in a cat) would also have to be in the right places too, which is still extremely unlikely. He is getting one thing right though: he is realising that knowledge that some of the cat exists makes it more likely that the rest of it exists.

The second mathematician also has a limited view. We will call him a *naive algorithmist*. He only considers the length of the partial model algorithm needed to detect the object or the length of an algorithm needed to describe or construct it. This makes any probability values that he computes naive algorithm probabilities. When faced with the same situation as above this mathematician can only assess the probability of the entire twenty bit sequence of bits occurring from the length of the algorithm needed to create it. For very long sequences of bits this could lead him to give higher probabilities than the naive objectivist might give, because some long sequences of bits might be described by very short algorithms: this mathematician is taking account of the way in which Occam's razor favours these. This does not make his view perfect, however. All he can think about is algorithm length. This means that he cannot distinguish between a situation where there is knowledge that part of the object exists and no such knowledge, because his view simply does not allow for examination of individual parts of objects. Confronted with half a cat he may conclude that the chance of this being part of a full cat is unlikely simply because cats are complicated. The algorithm needed to describe a cat is quite long: the significant knowledge that there is already evidence for half a cat cannot be incorporated into such a methodology, something which is dealt with by the naive objectivist who, unfortunately, does know about Occam's razor.

Neither mathematician really knows what is going on! Each of them, however, has a piece of it.

Why would we want to use naive probabilities?

If the naive objectivist or naive algorithmist were walking around in our world they would both appear insane. Why do I find their positions useful? It is not for any reason of disagreeing with you about their sanity.

Let us suppose that we must decide how likely it is that a particular object exists. Some observations of the place where the object is supposed to be can be made, but not enough to have certainty about whether or not it exists. In addition, we are not allowed to go and look ourselves. A naive objectivist and a naive algorithmist are going to go and look on

our behalf and come back to report their views to us. We are not naive, so we know not to accept the judgement of either of these mathematicians without question, but that does not mean it is useless to us. We can listen to what they both say and make our own judgement: the two types of naive information, combined together, can allow us to make a less naive assessment of probability.

This is how naive probabilities can be useful to us. The computation of naive object probabilities and naive algorithmic probabilities is easy to formalize, precisely because the consideration on which each type is based is so limited. This means it is easy to make algorithms to compute naive object probabilities and naive algorithm probabilities for the existence of objects. A further computation can combine the information from these two types of probability to generate *real* probabilities for the existence of objects.

Computation of Naive Object Probabilities

The decision tree is used to compute two naive object probabilities. These are:

- the basic object completion probability $P(\text{Basic})$
- the completion occurrence probability $P(\text{Completion})$

These two values are used to calculate two more naive object probability values:

- the evidence existence probability $P(\text{Evidence})$
- the misleading evidence probability $P(\text{Misleading Evidence})$

The Basic Object Completion Probability $P(\text{Basic})$

The basic object completion probability is the naive object probability of the existence of the object for a particular index in the matrices of bits when we have no information about the bit values in the matrices.

This is equivalent to the naive probability of a cat existing at a certain place when we have not looked.

The basic object completion probability is the probability value generated by collapsing the decision tree for the partial model when no knowledge is assumed about the matrices of bits. This means that wherever a node has two branches for reading “0” or “1”, each branch is assumed to have a probability of 0.5, *regardless of any probability values stored for the matrix.*

To generate the basic object completion probability $P(\text{Basic})$ we simply collapse the decision tree for the partial model, as previously discussed, but instead of using the matrices that we have in our hierarchy we use matrices in which every probability value is 0.5.

The basic object completion probability is a characteristic of the partial model algorithm only: it only needs to be computed once for any partial model.

The Completion Occurrence Probability $P(\text{Completion})$

The completion occurrence probability is the naive object probability of the existence of the object for a particular index when we have some probabilistically expressed information about the relevant bits in the matrices.

This is equivalent to the naive probability of an entire cat existing at a certain place when we have performed enough limited observation to see part of a cat.

To generate the completion probability $P(\text{Completion})$ we simply collapse the decision tree for the partial model, using the matrices of probability values that we have in our hierarchy.

The Evidence Existence Probability $P(\text{Evidence})$

The evidence existence probability $P(\text{Evidence})$ is a naive object probability that the amount of “evidence” processed by the partial model would exist.

The data in the hierarchy’s matrices which is read by the partial model algorithm for the current index can be considered to be “evidence” for the existence of the object that the partial model detects. Varying amounts of evidence can exist and the evidence existence probability $P(\text{Evidence})$ is the naive object probability that this amount of evidence would exist by chance without any process that happens to favour its creation occurring – that is to say, without assuming that the object is anything more than a random pattern.

This is equivalent to seeing half of a cat and computing the probability that this much evidence for a cat could exist *by chance* without any process that happens to favour the creation of cats generating it.

We do not need to do any more decision tree collapsing to obtain the evidence existence probability $P(\text{Evidence})$: we can compute it from the other naive probability values that we have already obtained. For the object to exist there must be *at least* the amount of evidence with which we are dealing – in some form – and the naive object probability of this occurring is $P(\text{Evidence})$. When this amount of evidence exists the probability of the object existing is the completion occurrence probability $P(\text{Completion})$, so if we know nothing about the values in the matrices, the object’s existence is based on the amount of evidence that we have existing and the object existing given this amount of evidence. This gives the following equation:

Equation 1: $P(\text{Basic})=P(\text{Evidence})P(\text{Completion})$

Rearranging this gives us the following equation that we can use to compute the evidence existence probability $P(Evidence)$:

$$Equation\ 2:\ P(Evidence) = \frac{P(Basic)}{P(Completion)}$$

The Misleading Evidence Probability $P(Misleading\ Evidence)$

The evidence existence probability $P(Evidence)$ is the probability that this amount of evidence could exist by chance without really meaning much. A problem with this is that, even if this amount of evidence does exist by chance, the remainder of the evidence needed to establish with certainty that the object exists could *also* exist by chance.

In discussing the evidence existence probability $P(Evidence)$ I gave the example of computing the probability that half of a cat could exist by random arrangement of matter that has nothing to do with cats. This is important to us because it has an impact on how likely it is that half a cat is part of a full cat, or just some lucky arrangement of matter. A complication is that even half a cat just exists randomly, then the other half of the cat could also exist randomly, thereby giving us a complete cat! We are only interested here in the probabilities of amounts of evidence existing that do *not* lead to complete objects. If we see half a cat then we need to compute the probability of half of a cat existing *and* the rest of the information that have yet to observe not randomly resulting in a complete cat. This value would be $P(\text{half a cat happens to exist randomly})(1 - P(\text{given half a cat, the other half of a cat happens to exist randomly}))$

We therefore need to make a new naive probability, the misleading evidence probability $P(Misleading\ Evidence)$. This is the probability that the amount of evidence which we have for the existence of the object would exist randomly, without any consideration of Occam's razor, without being part of a complete object, meaning that the remaining amount of evidence needed to satisfy the partial model algorithm does not randomly exist without any consideration of Occam's razor. This value is computed as follows:

$$Equation\ 3:\ P(Misleading\ Evidence) = P(Evidence)(1 - P(Completion))$$

We could actually get this value directly from the previous naive probability values by using a single equation:

$$Equation\ 4:\ P(Misleading\ Evidence) = \frac{P(Basic)(1 - P(Completion))}{P(Completion)}$$

which can be simplified to:

$$Equation\ 5:\ P(Misleading\ Evidence) = \frac{P(Basic)}{P(Completion)} - P(Basic)$$

Computation of a Naive Algorithm Probability: the Algorithm Satisfaction Probability $P(\text{Algorithm Satisfied})$

Whereas the naive object probability values were computed from the matrices of probability values, with reference to the partial model only to make the decision tree, a naive *algorithm* probability is computed only from the partial model algorithm, with no reference to the matrices in the hierarchy - even within the context of hypothetical execution of the partial model algorithm.

Just one naive algorithm probability is computed. This is the probability of algorithm satisfaction $P(\text{Satisfied})$. It depends on the length of the partial model algorithm (measured in bits) and is computed as follows:

Let n = the number of bits in the partial model algorithm.

Equation 6: $P(\text{Algorithm Satisfied}) = 0.5^n$

Shorter algorithms give a higher value for the algorithm satisfaction probability. The above equation can be considered a formal statement of Occam's razor, with the qualification that this probability value is a naive algorithm probability. Occam's razor is not telling us how likely it is that an object exists: it merely generates one of a number of naive probabilities that contribute to the result. The role of Occam's razor is still critical, however.

Combination of the Naive Probabilities

When we have the misleading evidence existence probability $P(\text{Misleading Evidence})$ and the probability of algorithm satisfaction $P(\text{Algorithm Satisfied})$ we can combine these values to obtain the actual probability that the object exists $P(\text{Object Exists})$ at that index, meaning the probability that the partial model algorithm being considered would indicate a successful object detection for that index if all the actual bit values in the matrices were known.

The probability that the object exists $P(\text{Object Exists})$ is computed as follows:

Equation 7:
$$P(\text{Object Exists}) = \frac{P(\text{Algorithm Satisfied})}{P(\text{Algorithm Satisfied}) + P(\text{Misleading Evidence})}$$

We could also express this as follows:

Equation 8:
$$P(\text{Object Exists}) = \frac{P(\text{Algorithm Satisfied})}{P(\text{Algorithm Satisfied}) + \frac{P(\text{Basic})}{P(\text{Completion})} - P(\text{Basic})}$$

which can be expressed as:

Equation 9:

$$P(\text{Object Exists}) = \frac{P(\text{Algorithm Satisfied})P(\text{Completion})}{P(\text{Algorithm Satisfied})P(\text{Completion}) + P(\text{Basic}) - P(\text{Basic})P(\text{Completion})}$$

or:

$$\text{Equation 10: } P(\text{Object Exists}) = \frac{0.5^n P(\text{Completion})}{0.5^n P(\text{Completion}) + P(\text{Basic}) - P(\text{Basic})P(\text{Completion})}$$

Notes on the result

- The maximum probability of an object's existence is 1 (that is to say, certainty). This is always the case when $P(\text{Completion})$ is 1. This happens when, given the probability values in the matrices which are being read by the partial model algorithm, no path exists through the decision tree which results in a failed object detection. This can only happen when at least bit that is read by the partial model algorithm is 1 or 0 (otherwise alternative paths through the decision tree, however unlikely, would still be possible) or when the partial model algorithm always indicates success in object detection (which would make it a trivial partial model).
- The minimum probability of an object's existence is 0 (that is to say, impossibility). This is always the case when $P(\text{Completion})$ is 0. This happens when, given the probability values in the matrices which are being read by the partial model algorithm, no path exists through the decision tree which results in a successful object detection. This can only happen when at least one bit that is read by the partial model algorithm is 1 or 0 (otherwise alternative paths through the decision tree, however unlikely, would still be possible) or when the partial model algorithm always indicates failure in object detection (which would make it a trivial partial model).
- The probability of an object's existence increases as $P(\text{Algorithm Satisfied})$ increases – in keeping with Occam's razor as a short algorithm implies a high value for $P(\text{Algorithm Satisfied})$.
- The probability of an object's existence increases as $P(\text{Basic})$ decreases. This may seem strange to some readers because it may appear to be suggesting that an object is more likely to exist when it is unlikely object! In a way, this is true, but the contradiction only *seems* to occur when we mix consideration of conventional and naive probabilities. When we have partial evidence of an object then the full pattern corresponding to the object is more likely to exist if the partial information that we have is not just a random occurrence and it is less likely to be such a random occurrence when $P(\text{Basic})$ is low.
- The probability of an object's existence increases as $P(\text{Completion})$ increases.
- It is wrong to presume that a low value for $P(\text{Algorithm Satisfied})$ is automatically equivalent to an indication of how likely it is that the partial model will be

- “correct”. I emphasize again that $P(\text{Algorithm Satisfied})$ is a naive probability. Only the final result from the equation really tells us anything about probability.
- Given that the probability of an object’s existence increases when $P(\text{Algorithm Satisfied})$ and $P(\text{Completion})$ increase and $P(\text{Basic})$ decreases, this tells us that we should most expect an object to exist when the algorithm describing it is short, when the object would naively be considered unlikely without any prejudice from Occam’s razor and when there is a lot of evidence for the existence of the object. This is a rather vague way of describing this as it does not explain what evidence is or how we evaluate it: the actual processes that I described earlier does this better.

More Naivety

There are other influences on the probability of an object’s existence that are not dealt with in this article.

The probability that an object exists is also influenced by its *frequency of occurrence*. If we know that an object happens to occur very often then this will have an influence on the probability that we assign to its existence in individual situations.

The probability of an object’s existence is also influenced by *context dependent frequency*. The context, or situation, in which some information exists can influence its probability: some objects are more common in certain situations. As an example, if we see part of an animal in someone’s house then we are usually more likely to conclude that it is a domestic cat than a lion because, without any reference to decision trees or partial models describing cats we know that cats are more common in houses than lions. If we applied the process in this article and it gave the same probabilities for the existence of a cat and the existence of a lion in identical situations in houses we would probably say that the cat probability should be adjusted to reflect the fact that people tend to put them in houses more often than lions, thereby skewing the statistics. Reality is full of things that can skew the statistics and most of these need not be caused by people as in my example.

Frequency of occurrence is just a special case of context dependent frequency in which we make the “context” expansive to the point where it includes any situation and we can ignore the context itself and just talk about frequency.

The omission of any consideration of context dependent frequency from the process described in this article means that there is still some naivety in the final probability that it provides. Similarly, if we did not use the process in this article and just obtained our probability from context dependence then we would also obtain a naive probability value.

This is not a big problem. This article deals with the main issues and describes the main part of the process that is needed. We have already been using results from processes which generate different naive probabilities, so we should be able to extend the process to do this again. We will need to add another process, *downward transfer of probabilities*, which I will briefly discuss shortly.

For now, the probability values generated by the process are probably quite usable in many applications. “Naive” does not mean unusable. It simply refers to a lack of some information when the probability value is being computed. All probability values describe lack of information. From the point of view of someone who had some kind of prescient knowledge of the outcome, all probability estimates by anyone lacking such knowledge would be naive. This also applies in situations where someone has better knowledge than us. Here is an example:

As far as we are concerned, when a ball has been placed on a roulette table it is equally likely to land on any number. This is not really saying anything about the roulette ball, however: it is saying something about limitations in our knowledge. In practice, the eventual resting position of the ball has been determined by its position, speed, the rate of rotation of the table and other factors. If someone could obtain better information about the situation and quickly do some computations he/she might have better information about the outcome and have some idea on which numbers the ball is more likely to land, maybe even to such an extent that he/she can place bets and expect to win. This is not necessarily just a thought experiment: there are claims of this being done in roulette by people using various devices to record the state of the table and the ball and hidden computers to predict the outcome [13,14]. To someone able to use such a method, the equal probability that we assign to each number on the roulette table would seem naive.

I have previously been treating naive probabilities as abstractions and the final result as real, but the naive probabilities in this article are really just probabilities computed with the same limitations on knowledge that are implied by any probability value except 0 or 1. Some readers may argue that there is a difference with these naive probabilities because the calculations are *wrong*, but the calculations are not wrong: they are simply incomplete. “Incomplete” simply means that there is some extra information that we could add to our computations – in the form of a useful computational method – that we do not know about. *Any* probability, however, is simply a result of incomplete information of some kind: lacking information about extra features we can add to an algorithm is ontologically no different from lacking information about features of reality. In this sense, all probability values which express any uncertainty (any values except 0 or 1) are naive.

Although there may not be any ontological difference between the naive probability values discussed in this article and any other values this makes the term “naive” so general as to be worthless. I would prefer the term to be useful within the context in which I am using it, so it may be helpful, in articles like this, to define a “naive” probability as one for which we acknowledge limitations in the particular *computational method* used to obtain it.

I have discussed all this to show that, although there is still some naivety in the type of probability value returned by this process, this does not make it ontologically different to any other probability value and it does not make it wrong. From a practical point of view, however, lack of information about computational methods that can be used in probability calculations is likely to have greater effect than lack of information about

features of reality because it will probably more generally compromise probability calculations.

Although the process may work without considering context dependent frequency, humans do consider it and some of the probability values that such a process gives may appear strange by human standards. We should still want to use any available computational method to improve the probability estimates and context dependent frequency will be dealt with by *downward transfer of probabilities*.

Downward Transfer of Probabilities

Downward transfer of probabilities is a way of dealing with *context dependent frequency*. Some objects will be more or less common in some situations. This needs to be reflected by a modification to the process to deal with this.

Downward transfer of probabilities will involve higher layers of the hierarchy influencing the probabilities of lower layers. The current process only involves probability information being passed up the hierarchy, but downward transfer of probabilities will also involve probability information being passed down the hierarchy so that the probability that an object exists is, to some degree, affected by meanings that have been extracted higher up in the hierarchy.

Downward transfer of probabilities will be added to the process in a later article.

Limitations on Computation

The process described here would be limited by computational resources. Limits are likely to be imposed on the lengths of partial model algorithms and the depths of decision trees.

The Lengths of Partial Model Algorithms

This article has described how to use partial model algorithms to assign probabilities to objects, but not how we would actually make them. Whatever process we use to make the partial model algorithms, NP-hardness – a very rapid increase in computational resources and time needed – is likely to be an issue. We have to somehow search the set of possible partial model algorithms to find ones that we want and unless some new, efficient method becomes available – and we should assume that it will not when designing the system – as the algorithm lengths increase the size of this set increases and the task of finding partial model algorithms that fit whatever criteria we are using to select them becomes harder.

We will probably only be able to generate *very* short partial model algorithms, maybe even of only a few bits each, and it is important that anyone who has the criticism of NP-hardness in the process that I have described understands this: I do not expect unfeasible things from computers.

This limitation should not worry us unduly: humans are probably subject to the same kind of limitation. We do have a partial solution to the problem and it is already built into the system. The hierarchical approach allows us to spread a complex meaning extraction over many levels of a hierarchy. The individual partial model algorithms performing meaning extraction can be small, and can be feasibly generated, but the meaning that they extract can be used by partial model algorithms higher up in the hierarchy to extract more complex meanings.

The Depths of Decision Trees

Further difficulties are caused by the use of partial model algorithms after they have been created. Partial model algorithms need to be used to make decision trees. As a decision tree goes deeper more branches will usually spread out and more computational resources or time will usually be needed to construct such a decision tree or to collapse it to a probability value. This means that a partial model algorithm, even a short one, which reads a large number of bits from the matrices could have an unmanageable decision tree. It may seem that partial model algorithms must always read only a small number of bits from the matrices from which they are extracting meaning.

As with the lengths of partial models, the main solution is provided by the hierarchical approach. Although only a small number of bits may be read by a single partial model algorithm applied for some index, partial model algorithms higher up in the hierarchy can combine the meanings extracted by multiple partial models for multiple indices.

It would be useful, however, if a single partial model algorithm, extracting meaning for a particular index value, could read a large number of bits and determine whether or not they fit the pattern of an object. Reading large number of bits does not necessarily imply the high complexity of long partial model algorithms: a simple relationship could be described between each of a very large number of bits. Can this be done within reasonable computational limitations?

There are two ways in which this could sometimes be possible:

First, the reading of a large number of bits does not necessarily imply that the decision tree becomes very wide and unmanageable. If many of the routes down the tree terminate very quickly then this will limit the width of the tree. This could happen with a partial model which reads a large number of bits and, right after reading each bit, checks if some condition is met, terminating with a failed object detection if it is not, with few possibilities being permitted for which termination does not occur. As an extreme example, an algorithm to check that each of a very large number of bits was “1” would not be a wide decision tree: there would only be one route through it which did not result in immediate termination.

Second, if a partial model algorithm is reading data about which there is little uncertainty – that is to say, if the probability values read by the partial model are almost all very close to 0 or 1 – then, although there are many possible routes through the decision tree, only a

small number of these would be likely. It may be possible to remove low probability routes through the decision tree, in a way analogous to forward pruning in chess algorithms [15,16]. This analogy is not perfect: forward pruning in chess is based on the expected undesirability of the outcomes below a particular node in the decision tree, whereas the forward pruning here would be based on the low probability of outcomes below a node.

Generation of Partial Models

This article has described the use of partial model algorithms to assign probabilities to objects, rather than how to make the partial model algorithms. This will need to be dealt with later.

One way of generating partial models could simply be to generate short sequences of bits, corresponding to partial model algorithms, randomly. The partial model algorithms produced in this way would be tested according to some criteria to determine their usefulness. This would only work for very short algorithms, but small algorithms could be produced in this way; for example, if partial models were all 30 bits long then there would only be about a billion different possible partial model algorithms. The hierarchical system would make up for the inability to have long algorithms.

An improvement would be if we could find a better way of generating partial models than trying random algorithms. One possibility could be to use some kind of genetic technique. It should be noted that such techniques will not generate every possibility and, just as when we ascend the hierarchy, there is the possibility that we may fail to extract meanings that the system simply does not find.

A further way of expressing the algorithms is to reduce them to some simple system. Rather than let each partial model run as a low level program, which could make it hard to get much into a small program size, we might have a standard algorithm design and use the bits in the partial model “algorithm” just to define parameters for it. This could make the system closer to a neural type system as the standard algorithm may make all partial algorithms act like simple switching units. There is the possibility of losing the capability to express all meanings here, however, so caution is needed.

Actions

The process described in this article deals only with recognition of patterns by computers to make worldviews. For a general purpose artificial intelligence system some way for the system to use worldviews to plan actions is needed.

Having probability as an integral part of a hierarchical ontology, as it is in this process, is important. This is because the act of making a decision can be viewed as reducing uncertainty in a system’s own future behaviour and, for a system to model its own behaviour properly, it must actually model behaviour about which it is uncertain – the degree of uncertainty depending on the extent to which various decisions have been

made. A probabilistic description of reality will therefore be needed by a system to model the uncertainty in its own behaviour.

I will expand on this in later articles.

The Carpet Texture Problem

The carpet texture problem occurs when a system intended to find patterns for some purpose cannot distinguish between important and irrelevant patterns. It gets its name from the idea of a computer that is supposed to do some important task spending huge processing resources on finding patterns in a carpet.

The carpet texture problem can manifest itself in this process by computational resources being wasted in two ways:

- on *generating* partial model algorithms that are not needed.
- on *using* partial model algorithms to extract meaning in situations in which the meaning is of no interest.

The process as described in this article contains nothing – yet – to resolve the carpet texture problem and a solution will be needed later. I think that the capability of the system to plan actions is important in helping to *define* what computations are relevant.

I think that the feeding down of planned actions from upper levels of the system will play a role in resolving the carpet texture problem: actions selected by higher levels in the system should determine how computational resources are expended at lower levels.

Dimensions, Matrices and Indices

I am not entirely happy with the system of using indices which I have proposed to differentiate between different applications of partial model algorithms. I think that a more general solution will probably be needed. Any aspect of this ontology is subject to alteration later. Some way of differentiating between partial model algorithm applications is clearly needed and the method of using indices is intended as a prototype way of doing this.

One problem is that it assumes that the index is the *only* way in which different partial model algorithm applications can be differentiated. There will be other ways of differentiating between partial model applications and saying that the same basic pattern is found by two different runs of an algorithm.

It should be noted that the way in which indices are used by partial models is somewhat artificial and reflects a convenient way of viewing reality for humans and machines. The truly basic position, given a hierarchy, would be one of *no* indices, nor anything else to differentiate between different applications of partial models. This would not be a description of reality that would be of any practical use to us or machines. As an

example, the idea of having a concept of “a cat” which can be applied in different ways (using different indices for the partial model algorithm or, more realistically, for an entire hierarchy of partial model algorithms that play a part in “detecting” cats) would be meaningless. Every instance of what we would call “a cat” in reality would need its meaning extracting by a different partial model and the term “cat” would be meaningless: there would be nothing to unify these disparate partial models. Having the ability to make different applications of the same partial model allows a general concept of a thing – which we need.

A further problem I have with the way in which my indexing system works is that it requires all the matrices to have the same number of dimensions. The number of dimensions in the matrices is also a matter of convenience. A system like this would actually work, in principle, if all the matrices were one-dimensional, so why bother with more dimensions at all? Although meanings could be extracted from one-dimensional matrices, extraction of some meanings would be particularly simplified from matrices with particular numbers of dimensions. The process that I have proposed goes some way to achieving this, by acknowledging the idea of matrices with any number of dimensions, but it does not provide a workable solution because there is no way for meaning extraction algorithms to produce matrices with numbers of dimensions different to those from which they extract their meaning. I think that extending the process to allow such *dimensionalization* would be useful.

All of this means that the way in which the indices and matrices work, and in which different partial model algorithm applications are differentiated from each other, needs to be improved to provide more flexibility. I have proposed the system as it is now because I needed *some* way of expressing it to get the rest of the system in place.

Conclusion

This article has discussed how a hierarchical modelling system could work in which meaning is extracted by *partial model algorithms* or *meaning extraction algorithms*.

A partial model algorithm extracts meaning from matrices of bits and takes the form:

$$R_i = M_i(P_i, B_1, B_2, \dots, B_n, i)$$

where:

M_i is a meaning extraction algorithm – the partial model algorithm itself.

R_i is the result returned by the partial model algorithm for some index i . For the partial model algorithms that we are considering, this result will be “1” for successful object detection or “0” for failed object detection.

P_i is a parameter without any purpose at the moment, but included in case it is needed later.

i is a vector describing a location in a matrix or matrices.

B_1, B_2, \dots, B_n are the matrices of previously extracted meaning on which the partial model operates.

The result R_i generated for different values of the index i implies a new matrix and other, higher level partial model algorithms can extract meaning from it.

The types of partial models that are relevant to this article are *object-like* partial models which can be viewed as returning an object detection result for any index – “1” for successful object detection or “0” for failed object detection.

The results from a partial model algorithm for different indices imply a new matrix of extracted meaning from which further meaning can be extracted by higher level partial model algorithms.

Although this description is in terms of partial model algorithms extracting meanings in the form of bits and extracted meanings being stored as bits, the data actually stored about reality does not take the form of bits: it is in the form of probabilities of bit values. This uncertainty appears because, in a real situation in which a computer is extracting meanings from sensory inputs, only some of all possible inputs that could occur are known. Any meaning that is extracted, wholly or partially from unknown bits must be represented by a probability of the meaning extraction algorithm returning a successful object detection.

The approach works by inferring probabilities of bits being “1” in higher level matrices, given that the values of the bits in those matrices are a result of meaning extraction from lower level matrices and, ultimately, matrices of bits obtained directly from sensory input. The process therefore infers all of the probabilities in the hierarchy from these initial inputs, and these are expressed in probabilistic terms. If a sensory input has a value of “1” then a probability of 1 is stored for it in the hierarchy’s matrices. If the value is “0” then a probability of 0 is stored. If the value is unknown, because the input has not occurred yet, then a probability of 0.5 is stored. The “bottom level” of the hierarchy, therefore, contains probabilities from which all the other probabilities are inferred.

Each time a partial model algorithm is “applied” in a real situation, we do not literally apply it to the data. Instead we perform a computational process to determine the probability that the partial model algorithm would return a successful object detection for a particular index.

The calculation involves the use of a *decision tree* - a structure describing every possible route through the different states of a partial model algorithm until it returns success or failure in object detection. The nodes at which the decision tree terminates have probability values of 0 or 1. In each case the value used is numerically identical to the result returned by the partial algorithm for that node, but they are not the same thing. The results returned by partial model algorithms are bits. The value placed at a terminating node of a decision tree is a *probability* of such a bit being “1”, which will be propagated up the decision tree and used to generate other probabilities.

A recursive process generates the decision tree for a partial model algorithm. This only needs to be done once for a particular partial model algorithm: the decision tree is not specific to any index used.

The decision tree can be collapsed to generate a probability value. This can be done for a particular index or any set of assumed matrices of probability values describing what is known (possibly hypothetically) about meaning extracted at lower levels.

The calculation involves generation of *naive probability* values which are combined to generate the final probability of successful object detection by the partial model. Naive probability values are computed with what we would regard as an omission in the computational process used to obtain them. Two kinds of naive probability values are relevant here: *naive object probabilities* and *naive algorithm probabilities*.

The naive object probability calculations are as follows:

The basic object completion probability $P(\text{Basic})$ is calculated by collapsing the decision tree for the partial model. The actual matrices for meanings previously extracted from reality are not used. Instead, the collapse is performed with the assumption that any matrices from which bits are read contain only probability values of 0.5.

The completion occurrence probability $P(\text{Completion})$ is calculated by collapsing the decision tree for the partial model algorithm with the assumption that the matrices from which the partial model algorithm is extracting its meaning are those matrices of probabilities obtained by previous meaning extraction – that is to say, the matrices that are part of the model.

The evidence existence probability $P(\text{Evidence})$ is the naive object probability that the amount of evidence on which we based any expectation of the partial model algorithm returning successful object detection would occur at this particular index and is calculated as follows:

$$P(\text{Evidence}) = \frac{P(\text{Basic})}{P(\text{Completion})}$$

Equation 2:

From these values the misleading evidence probability $P(\text{Misleading Evidence})$ is computed as follows:

$$P(\text{Misleading Evidence}) = P(\text{Evidence})(1 - P(\text{Completion}))$$

Equation 3:

We could get this value directly from the previous naive probability values and miss out the calculation of $P(\text{Evidence})$ by using a single equation:

$$\text{Equation 4: } P(\text{Misleading Evidence}) = \frac{P(\text{Basic})(1-P(\text{Completion}))}{P(\text{Completion})}$$

A single naive algorithm probability is computed: the probability of algorithm satisfaction $P(\text{Satisfied})$. It depends on the length of the partial model algorithm (measured in bits) and is computed as follows:

Let n = the number of bits in the partial model algorithm.

$$\text{Equation 6: } P(\text{Algorithm Satisfied}) = 0.5^n$$

The naive object and algorithm probability values are now combined to compute the final result - the probability of object existence $P(\text{Object Exists})$ - as follows:

$$\text{Equation 7: } P(\text{Object Exists}) = \frac{P(\text{Algorithm Satisfied})}{P(\text{Algorithm Satisfied}) + P(\text{Misleading Evidence})}$$

$P(\text{Object Exists})$ is the probability that the partial model algorithm would indicate successful object detection if all of the unknown bits in the matrices corresponding to sensory inputs had known values, causing the values of all the other bits in the matrices to be known.

We could also express this as follows:

$$\text{Equation 8: } P(\text{Object Exists}) = \frac{P(\text{Algorithm Satisfied})}{P(\text{Algorithm Satisfied}) + \frac{P(\text{Basic})}{P(\text{Completion})} - P(\text{Basic})}$$

which can be expressed as:

$$\text{Equation 9: } P(\text{Object Exists}) = \frac{P(\text{Algorithm Satisfied})P(\text{Completion})}{P(\text{Algorithm Satisfied})P(\text{Completion}) + P(\text{Basic}) - P(\text{Basic})P(\text{Completion})}$$

or:

$$\text{Equation 10: } P(\text{Object Exists}) = \frac{0.5^n P(\text{Completion})}{0.5^n P(\text{Completion}) + P(\text{Basic}) - P(\text{Basic})P(\text{Completion})}$$

The result is a probability and a different probability is obtained for each different index for which the partial model algorithm can be used to extract meaning. These probability values can be used to form a new matrix of probabilities – the matrix of meanings implied by this partial model algorithm – which becomes part of the hierarchy and can have meaning extracted from it by still higher level partial model algorithms.

This equation is intended to be used for partial model algorithms to generate meaning hierarchically in artificial intelligence, but the same idea could also be used to test any claim that can be formally expressed, at least in principle. To do this we would make an algorithm which defines that claim. It would do this by producing successful object detection results when it detects the claim's "truthfulness" and failure otherwise. It should be noted that should such an approach become common then arguments about *triviality* of various claims are likely.

Downward transfer of probabilities will be discussed in a later article. This is a way of dealing with *context dependent frequency*. Some objects will be more or less common in some situations. This needs to be reflected in a modification to the process to deal with it. Downward transfer of probabilities will involve higher layers of the hierarchy influencing the probabilities of lower layers. The need for downward transfer of probabilities to be taken into consideration indicates that there is still some naivety in the final probability results generated by this process. This does not mean that the probability values are wrong: they would probably be usable in many practical situations. All probability values describe lack of knowledge and a naive probability value is simply one for which the lack of knowledge relates to computational techniques for calculating the probability. Naive probability values are ontologically no different from any other probability value.

This article has not discussed in detail any method for generating partial model algorithms: it has dealt, instead, with how a description of reality can be represented and how probabilities can be assigned in the hierarchy. An artificially intelligent system would need some way of generating the partial model algorithms automatically, according to some criteria, and this will be discussed in later articles. It should be noted that the description in this article is very general and a working system would not necessarily have to do *everything* in this article: partial model algorithms may be represented by a set of bits, for example, that serve as parameters for some small, basic meaning extraction algorithm, so that the same basic algorithm is used for every partial model with the parameters making changes each time. A system that has similarities with neural systems might be created in this way.

The probabilistic way of representing the ontology is particularly important with regard to planning actions and the carpet texture problem. I think the concept of planning actions in a hierarchical model is only really coherent when the model is expressed in probabilistic terms, as working with a probabilistically expressed model of its own future behaviour is part of the planning process. Planning starts with the system being uncertain about its own future actions and involves reducing this uncertainty. This requires the model to be able to incorporate uncertainty. I think that a probabilistically expressed ontology is also important to the carpet texture problem because planning plays a key role

in this by providing a definition of what is relevant. All of this will be discussed in later articles.

Available computational resources will limit what can be done with such an ontology. Extraction of meaning by using decision trees is NP hard and generation of partial model algorithms will almost certainly be NP hard. The main way of dealing with this is to use the hierarchy itself and limit the size of partial model algorithms and the depths of decision trees, so that only a small amount of meaning is extracted in each layer of the hierarchy. An approach similar to forward pruning in chess algorithms could allow increased depth in some cases in which there is low uncertainty.

Further issues to be resolved involve indices and dimensionalization. The way that indices are used with matrices is probably too simplistic. It was introduced to allow applications of a single partial model algorithm to be distinguished from each other, which is necessary to have anything like a general concept of an object in human ontology, but a more general system of distinguishing between different partial model applications may be better. The way in which indices are used does not allow matrices to be produced with different numbers of dimensions than those from which their meanings were extracted. These matters will need to be addressed in later articles.

References

[1] Web Reference: Almond, P. (2005). *Occam's Razor Part 1: What Is Occam's Razor?* Retrieved 22 August 2005 from <http://www.paul-almond.com/OccamsRazorPart01.htm>.

[2] Web Reference: Almond, P. (2005). *Occam's Razor Part 2: Principles of Language*. Retrieved 9 October 2005 from <http://www.paul-almond.com/OccamsRazorPart02.htm>.

[3] Web Reference: Almond, P. (2005). *Occam's Razor Part 3: Assumptions About Reality*. Retrieved 13 November 2005 from <http://www.paul-almond.com/OccamsRazorPart03.htm>.

[4] Web Reference: Almond, P. (2005). *Occam's Razor Part 4: An Overview of How Occam's Razor Works*. Retrieved 24 December 2005 from <http://www.paul-almond.com/OccamsRazorPart04.htm>.

[5] Web Reference: Almond, P. (2006). *Occam's Razor Part 5: How Mapping Can Work*. Retrieved 14 January 2006 from <http://www.paul-almond.com/OccamsRazorPart05.htm>.

[6] 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>.

[7] 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>.

- [8] Web Reference: Almond, P. (2005). *What is a Low Level Language?* Retrieved 17 July 2005 from <http://www.paul-almond.com/WhatIsALowLevelLanguage.htm>.
- [9] Hawkins, J., Blakeslee, S. (2004). *On Intelligence*. New York: Henry Holt.
- [10] 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>.
- [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] Bass, T. (1990). *The Newtonian Casino*. London: Longman.
- [14] Bass, T. (2000). *The Eudaemonic Pie*. USA: Universal Publishers.
- [15] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 4, pp59-60.
- [16] Heinz, E, A. (2000). *Scalable Search in Computer Chess: Algorithmic Enhancements and Experiments at High Search Depths*. Vieweg Verlag. pp2-3,24-26,29-61.