

A Proposal for General AI Modeling

By Paul Almond, 10 April 2009

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

© Copyright Paul Almond, 2009. All Rights Reserved.

A Proposal for General AI Modeling

By Paul Almond, 10 April 2009

1 Abstract

A probabilistic, hierarchical modeling system is proposed which generates a hierarchy of patterns, each with a probability. Lower-level pattern (or object) probabilities are generated from the data to be processed and used to generate still higher-level probabilities, and so on. High-level pattern probabilities also influence low-level ones, ultimately causing low-level, unknown data to be assigned probabilities. Meta-patterns are also used, providing a more general ontology by allowing description of the occurrence of the same kinds of patterns in different contexts and at different levels of the hierarchy. The modeling approach is intended for use in planning as modeling, an approach to AI planning which involves almost all planning being performed by a modeling system. When it is used in this way, predictions of future evaluation function scores could be obtained from the system to evaluate any given output. To make development of the modeling system more feasible, this article describes a modeling system intended to fill in missing parts of images. Such a modeling system could be adapted for use in general AI, in planning as modeling, later.

2 Introduction

In previous articles [1] I discussed an approach to artificial intelligence (AI) called *planning as modeling*. Planning as modeling needs a capable modeling system to analyze previous inputs, outputs and evaluation function scores and generate predictions of future evaluation function scores. I have suggested an approach to modeling previously [2,3,4,5], but this is an attempt to propose a system closer to something that could practically be implemented.

The proposed modeling system is *probabilistic* and *hierarchical*. The data to be processed is placed in the bottom level of the hierarchy and used, with statistics obtained previously for *patterns* (or objects), to generate probabilities for patterns at higher levels. These probabilities are used to generate still higher-level pattern (or object) probabilities, and so on. High-level pattern probabilities also influence low-level pattern probabilities and so the high-level patterns ultimately influence low-level predictions of future input events and evaluation function scores. This allows predictions of future evaluation function scores to be obtained from the system to evaluate any given output.

The proposed modeling system also uses *meta-patterns*. A meta-pattern is like a pattern, except that it describes a group of patterns. Meta-patterns are proposed to give a more general ontology, by allowing expression of the occurrence of the same kinds of patterns in different contexts and at different levels in the hierarchy.

The amount of information processed and stored needs reducing and a method of doing this will be suggested. Information should be removed from the hierarchy as time passes, so that very detailed information about the past does not need to be stored. A method for doing this will be suggested, based on the idea of removing lower levels of data from the hierarchy corresponding to the past and removing progressively higher levels of information further back into the past.

A modeling system for planning as modeling may present some complications, so as an intermediate step the modeling system described here is intended to solve a simpler problem – filling in missing parts of images. A modeling system for general AI use, and specifically planning as modeling, would work on the same general principles, but may have some differences.

(See Figure 1)

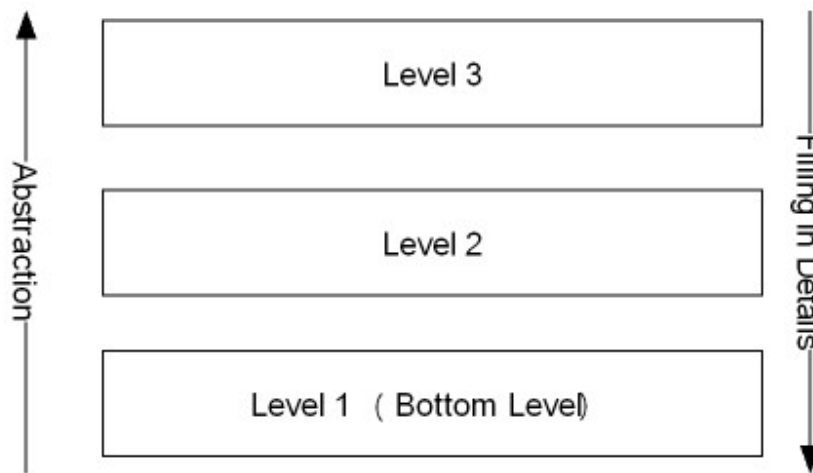


Figure 1: A Probabilistic, Hierarchical Modeling System

3 The Objective: Processing Images

As an intermediate step to getting a working modeling system for use in general AI in planning as modeling, I am proposing a system for use in image analysis.

The modeling system proposed here is to analyze an image which consists of an array of pixels, each pixel being black or white: that is to say, the modeling system is to analyze two-dimensional (2D) arrays of binary (0 or 1) values. (Dealing with a grayscale is unnecessarily complicated at this stage and preparation of images in this way for AI systems is known [6].) The modeling system is to generate probabilities for pixels in missing parts of the image. Missing parts of the image could be due to removal of part of the image or could correspond to the unseen area around the edges of the image.

The modeling system is *probabilistic* and *hierarchical*. Known image data is treated as probabilities and used to generate probabilities for patterns (or objects) at progressively

higher levels of abstraction. These in turn influence probability values of patterns in the levels beneath them, ultimately causing probabilities to be assigned to values corresponding to pixels in “missing” parts of the image for which no data is available.

4 Patterns

The system uses layers of *patterns*. Each pattern is on a level. There are multiple patterns on a level. There are multiple levels. Patterns are identified by means of pattern identifiers. In the example given here it is assumed that each pattern identifier is a unique number.

Each pattern has a *pattern instance* for each (X,Y) coordinate. The pattern instance can be viewed as the output of the pattern. Each pattern instance is binary (0 or 1). For each pattern there is a 2D array of pattern instances.

All pattern instances for a pattern are generated by the same rules in the *pattern logic* for the pattern. The pattern logic could be expressed in a number of ways. The simplest is probably as a truth table. The pattern logic consists of information to identify a number of inputs into the pattern from other patterns, at lower levels in the hierarchy, and the logic that determines the state of an instance of the pattern from those inputs. If the pattern logic is based on truth tables, then the pattern logic for a pattern consists of the *input list* and a *truth table*. The input list of a pattern is a list of pattern instances of *other* patterns, each at an offset from the (X,Y) coordinate of the pattern which has this pattern logic. A *pattern identifier* and the offset are stated for each input in the input list. The pattern logic is a truth table indicating what the pattern instance is at any (X,Y) position, given the set of inputs from other pattern instances for that (X,Y) position.

Example: A simple pattern with two inputs from other patterns

Definition of Pattern 87

Pattern 87 Input List:

Input 1: Pattern 56, Offset (-2,1)

Input 2: Pattern 45, Offset (3,0)

Pattern 87 Truth Table:

Input 1	Input 2	Pattern Instance
0	0	0
0	1	1
1	0	1
1	1	0

Pattern 87 takes as inputs pattern instances from two patterns, Pattern 56 and Pattern 45, which must be lower-level patterns. These inputs are defined as being at particular offsets from the relevant pattern instance in Pattern 87.

Example of Application of Pattern 87 Logic

Suppose we want the pattern instance at position (20,35) for Pattern 87.

Input 1 is a pattern instance of Pattern 56, with offset (-2,1). For (20,35) this gives an absolute position of $(20-2,35+1) = (18,36)$. We therefore obtain the pattern instance from position (18,36) of Pattern 56. Suppose this is 0.

Input 2 is a pattern instance of Pattern 45, with offset (3,0). For (20,35) this gives an absolute position of $(20+3,35+0) = (23,35)$. We therefore obtain the pattern instance from position (23,35) of Pattern 45. Suppose this is 1.

The resulting pattern instance is looked up in the truth table. Input 1 is 0 and Input 2 is 1. This corresponds to the second line of the truth table which gives a result of 1. The pattern instance for position (20,35) of Pattern 87 is therefore 1.

Pattern Instances for Pattern 87 at other (X,Y) positions are determined similarly and a 2D array of pattern instances is generated.

A pattern instance at some (X,Y) position is only defined when the pattern instances referred to in its input list are defined. This means that at least one pattern must exist which is not dependent on other patterns. This purpose is served by a special pattern in Level 1 which is the bottom-level pattern. The pattern instances for the bottom-level

pattern are the actual binary (0 and 1) values of the image being processed. The bottom-level pattern has no pattern logic, as its pattern instances are already defined.

It may seem that most of the time will be spent computing truth tables, but the actual values for pattern instances will often be unknown and for much of the system's operation will be represented by probabilities.

There are alternatives to the use of simple truth tables for pattern logic. It may be desirable to have a large number of inputs, but with simple truth tables this would be impractical for random pattern generation as the chance of generating a useful pattern would be low. However, a compromise could be made in which the pattern's input list defines groups of inputs, each group of inputs existing within some region in another pattern. The truth table might refer to the most common inputs in groups of these inputs, instead of referring to individual inputs. Other variations on this kind of idea are possible. It might be that, instead of a truth table being used, the criteria for the pattern instance being 1 is stated in some other way. The idea of all this would be that a large number of inputs could be defined and used by the pattern, but they would be dealt with in groups in some way so that very little information was needed to define the inputs or the pattern logic.

Another alternative to simple truth tables would be to give patterns some similarity with neurons in artificial neural networks [7,8]. Each pattern could have a number of inputs and each input could be multiplied by a separate "connection strength" constant that could be positive (excitatory) or negative (inhibitory), to generate a value, the pattern instance being 1 if the sum of all these values exceeds a certain threshold and 0 otherwise. As with the previous suggestion, groups of inputs could be defined; for example a group of inputs may be defined in some region that all have the same connection strength. While neuron-like patterns may seem attractive, due to the apparent biological similarity and the previous success of neural network techniques, there should be a good reason for doing this, beyond seeking a possibly superficial resemblance to neural networks. Simple truth tables may be the best way of expressing pattern logic for now.

5 Processes Performed on the Hierarchy of Patterns

A number of operations are performed on the hierarchy of patterns. These are:

- Logic application
- Statistics generation
- Statistics application

I will now describe each of these operations:

5.1 Logic Application

This is the most obvious process. The truth table (or equivalent) for each pattern is used to generate the pattern instances for that pattern. The result is a pattern instance (a binary 0 or 1 value) generated with complete certainty for each (X,Y) position.

Logic application can only be performed for pattern instances that depend on known data. If, when generating a pattern instance for some position (X,Y) the pattern instances referenced by any of the pattern's inputs are undefined then the pattern instance at (X,Y) will itself be left undefined by the pattern logic application process. If this pattern instance is later referenced as an input during pattern logic application for another, higher-level pattern instance, then this pattern instance is in turn undefined and so on.

Pattern instances in the bottom-level pattern (the data from the image itself) will cause some pattern instances to be defined in the level above it and these will cause further pattern instances to be defined, and so on. However, the finiteness of the number of pattern instances in the bottom level will limit the number of pattern instances that get defined at higher levels. Only pattern instances which depend, directly or indirectly, only on *known* pattern instances in the bottom-level pattern will be defined. Pattern instances which depend, directly or indirectly, and wholly or partially, on any missing part of the image or on pattern instances which are off the edge of the known image will be left undefined by pattern logic application.

Logic application is not the main operation: it is a means to an end and its main purpose is to prepare data to be analyzed in *statistics generation* (below).

5.2 Statistics Generation

The purpose of statistics generation is to acquire statistical information about the frequency with which each of the possible combinations of inputs occurs for each pattern.

A large number of images should be available as a training set. For each image, the hierarchy would be initialized with the image data loaded into the bottom-level pattern instances. Logic application would then occur, as above, so that a hierarchy of pattern instances is constructed “above” the bottom-level pattern. This hierarchy is then analyzed to build up statistical information for each pattern. For each pattern, each pattern instance at a different (X,Y) position is processed to obtain the frequency with which each possible combination of inputs occurs. This data is acquired simply by examining pattern instances, following logical pattern application, and counting.

In the case of pattern logic being based on truth tables, this would mean analyzing each pattern instance at position (X,Y), in each training set image, for that pattern and counting the number of times each line of the truth table occurs. This kind of process would be best when the number of inputs is small and complications may result if truth tables with large numbers of inputs, or alternatives to truth tables, previously discussed, are used with large numbers of inputs.

The analysis occurs over a large number of images, with the same frequency values being updated. That is to say, the running totals are *not* reset after analysis of each image.

The statistical information (frequencies) computed in this way is stored as part of the pattern.

The main purpose of pattern logic application is to construct a hierarchy for use in statistics generation.

5.3 Statistics Application

Statistics application involves using the defined pattern instances, with the pattern statistics obtained from previous statistics generation to analyze an image and fill in missing parts of it.

In statistics application, pattern instances are dealt with as probabilities. Some processes occur before statistics application:

The statistics application stage is preceded by the placing of the image data into the bottom-level pattern instances. It is now treated as probabilities, so a value of 1 is considered as a probability of 1 and a value of 0 is considered as a probability of 0.

A logic application stage may follow in which the bottom-level pattern is used to generate the hierarchy of pattern instances implied by it according to the pattern truth tables or equivalent. What happens is the same as with conventional pattern logic application, discussed previously, but the values generated for pattern instances by pattern logic application are now regarded as probabilities of 0 or 1. The pattern logic application stage is optional – statistics application would generate probabilities for these pattern instances anyway – but it would reduce uncertainty in the hierarchy.

The statistics application process itself now works upwards to generate abstraction as follows:

For each pattern instance at position (X,Y) of each pattern (excluding the bottom level), the probabilities associated with the inputs for that pattern instance (which are probabilities for other pattern instances in lower levels) are examined and, in conjunction with the pattern statistics stored for that pattern, used to generate or modify a probability for this pattern instance at (X,Y).

It also works downwards, filling in details as follows:

For each pattern instance at position (X,Y) of each pattern (excluding the bottom level), the probability for that pattern instance is used, in conjunction with the pattern statistics stored for that pattern, to generate or modify a probability for each pattern instance that is an input for the pattern instance at (X,Y).

There is a conflict between different probability values because attempts will be made to update pattern instance probabilities after they have already been assigned. Statistics application must take account of this and further consideration will be given to this later. The issue of small, and possibly misleading, samples also needs addressing.

This process repeats until no further probability changes occur for pattern instances.

Initially, bottom-level pattern instances will be the only ones with probability values likely to change probabilities for other pattern instances, so application of the pattern statistics should start here and move to progressively higher levels, before moving down again. The idea is that low-level probability values influence higher-level probabilities, at greater levels of abstraction, which in turn influence lower-level probabilities.

Statistics application cannot affect bottom-level pattern instances, or any pattern instances which were derived, directly or indirectly, from them by pattern logic application as the actual values in these cases are already known.

The area over which (X,Y) coordinates are valid, and for which bottom-level pattern instances exist, is not limited to the area for which image data is known: it can be made as large as desirable. Some of the pattern instances for the bottom-level pattern will correspond to missing parts of the image. Statistics application will assign probabilities to these pattern instances. The system has therefore made predictions about the missing parts of the image. These predictions will have come from passing probabilities upwards from the bottom level to the highest levels of abstraction as generalizations are observed and downwards as details are filled in.

An important feature of statistics application is that it does not require much knowledge about all of the inputs for a pattern. For example, a pattern instance immediately above the bottom level can still be assigned a probability if only some of its inputs are known. Higher in the hierarchy, the dependence of pattern instances on known inputs becomes increasingly indirect. However, the parts of the hierarchy about which little is known become known in more detail when probabilities propagate downwards.

6 Pattern Generation

The system needs a library of suitable patterns. The simplest way of making patterns is random generation. Patterns are repeatedly generated in this way. When a new pattern is generated it is tested. The test involves experimentally adding it to the existing library of patterns and determining how well the system performs, with the training set, in determining probability values for bottom-level pattern instances. The best performing patterns are semi-permanently added to the system. The criteria for “best” might be that the pattern under test give(s) the best results obtained for the n previous patterns tested. For example, if n=1000 then the top $1/1000^{\text{th}}$ of randomly generated patterns are selected.

Random generation may not be limited to single patterns. A number of patterns might be generated and tested as a group. The number of patterns in such a group is likely to be small, however.

Limits may be imposed on random pattern generation. For example, offsets to coordinates may be limited in size or, if there are multiple patterns being used as inputs, they may be required to be within a given number of levels of each other in the hierarchy.

Although I have described random generation as the basic method, and the system should be viable with just that, further methods might be used. For example, patterns may be changed slightly, or new variations produced by copying existing ones and making small changes. Such a small change may be altering the offset of a pattern input. If patterns with groups of inputs are used then it could mean moving the boundary of the region containing a group of inputs. If patterns are made to act like neurons it could mean altering connection strengths. We might also imagine combining patterns. For now, random generation should suffice.

There is little chance of randomly generating any single pattern that achieves much. This is why a hierarchy is used. Low-level patterns which perform a small amount of useful abstraction can be generated and then used as a basis for further abstraction, and so on. Each subsequent level of patterns can benefit from the patterns beneath it already performing some useful abstraction on the bottom-level data.

As well as random pattern generation, random pattern removal may occur, with a randomly selected pattern being experimentally removed from the hierarchy to determine the effect that this has on modeling with the training set. This might be done because some patterns may become obsolete, for example, if they deal with special cases that are later encompassed by more general cases. (See the discussion on meta-patterns below.) If removal of a pattern were found to have negative results, the removal would be reversed. Removal of a pattern would also require removal of any patterns which used its pattern instances as inputs. (The possibility of building pattern removal into the system is the reason I said “semi-permanently” instead of “permanently”, above.)

7 Extending the Ontology: Meta-Patterns

The ontology as described so far is too simplistic and limited to do much. A weakness is that some generalizations which seem simple and obvious to humans would need to be expressed as a large number of patterns and this would need to be done exhaustively at many levels of the hierarchy. As an example, this is equivalent to a human having to spend weeks working out how to deal with seeing part of a collection of balls, arranged in a circle, and then having to spend more weeks working out how to deal with seeing part of a collection of stones arranged in a circle. Some types of pattern should be more general.

This is dealt with by *meta-patterns*. A meta-pattern is not itself directly involved in logical application, statistics generation or statistics application. Instead, it describes a set of patterns involved in these processes.

A meta-pattern's structure is like that of a conventional pattern, except that some of its information is replaced by variables. Different patterns are produced by giving the variables different values. A single meta-pattern could refer to a very large set of patterns, so limitations may be placed on meta-pattern generation to preclude generation of too many patterns. A meta-pattern can be thought of as a *pattern template*.

A simple sort of meta-pattern may accept inputs from a single pattern X, where X is a variable. This would cause a conventional pattern to be generated for every other conventional pattern in the hierarchy.

Example: Definition and use of a meta-pattern

Definition of Meta-Pattern 61

Meta-Pattern 61 Input List:

Input 1: Pattern X, Offset (0,1)
 Input 2: Pattern X, Offset (-1,-1)

Meta-Pattern 61 Truth Table:

Input 1	Input 2	Pattern Instance
0	0	0
0	1	1
1	0	1
1	1	0

and a conventional pattern is formed by substituting any pattern number for X.

Example of Use of Meta-Pattern 61

X=115

Pattern 124 Input List:

Input 1: Pattern 115, Offset (0,1)
 Input 2: Pattern 115, Offset (-1,-1)

Meta-Pattern 124 Truth Table:

Input 1	Input 2	Pattern Instance
0	0	0
0	1	1
1	0	1
1	1	0

A meta-pattern may have a combination of variable and fixed inputs that reference specific patterns. There may be limitations on the values that variables can take to form conventional patterns. For example, if there are two or more variable pattern inputs it may be specified that they must be within a certain number of levels of each other in the hierarchy.

A meta-pattern can be formed by making a pattern that references one or more meta-patterns as its inputs. Any pattern that references a meta-pattern as an input is itself a meta-pattern.

Meta-patterns are generated in the same way as conventional patterns. They are randomly generated and then tested, in use in the system. A meta-pattern is likely to add much more data processing to the system than a conventional pattern, by causing multiple conventional patterns to be generated.

During statistics generation, all conventional patterns associated with a meta-pattern are treated as if they are the same pattern. A single set of frequencies is maintained for all the patterns as a group: *a meta-pattern only has one set of statistics.*

Meta-patterns can be involved in a kind of recursion. A meta-pattern could be defined so that conventional patterns are created which refer to a set of patterns including those already created, using the meta-pattern, as inputs. The meta-pattern example that I just gave can be used to illustrate this. In the example, Meta-Pattern 61 had Pattern "X" as an input. Pattern 124 results when X=115, but a further pattern can be made using X=124, the pattern already generated by the meta-pattern. Whether or not the meta-pattern is applied to its own set of patterns like this should be stated in the meta-pattern description. If this happens there should be an upper limit on the number of conventional patterns that can be created.

The rules for meta-pattern construction might allow the conventional patterns created by it to refer to multiple patterns which have some relationship in common. For example, suppose that some meta-pattern, A, can be applied to any pattern, B, to make some pattern, C, and another meta-pattern, D, can be applied to B, to make some pattern, E. A meta-pattern could be defined which refers to any two patterns, C and E, which have been generated by meta-patterns A and D respectively, both of which acted with reference to the same pattern, B. The way that meta-patterns are encoded would need

some consideration: the more careful this consideration, the more general the ontology would be.

A *conventional* pattern is actually a kind of meta-entity: it refers to a set of pattern instances, each of which could have been dealt with as a special case by a separate entity. (In fact, there is the possibility that as well as generalizing with meta-patterns we make entities *less* general than conventional patterns, that only apply to pattern instances in a limited region. I will not discuss this in detail here, as I think it is not necessary right now and that it will be of little relevance to the image processing problem being considered.)

Use of meta-entities like this allows much greater efficiency, as a single act of learning, when a meta-entity is generated and found to be usable, can be used in many different parts of the hierarchy.

8 Reducing the Amount of Computing

*As I was going up the stair
I saw a man who wasn't there
He wasn't there again today
Oh, how I wish he'd go away...*
- *Antigonish*, William Hughes Mearns (1875-1965)

The issue in the above poem will become a real problem, unless it is resolved. This is about making sure that things that are not there “go away” – in the sense that we do not need to deal with them explicitly in the modeling system.

Any sophisticated system will need many patterns. Meta-patterns could cause generation of a huge number of conventional patterns. For example, a single meta-pattern could cause a conventional pattern to be generated for every other conventional pattern in the system or, for a meta-pattern with two variables corresponding to pattern identifiers and no restriction on the value that each can take, for every combination of two patterns. Each pattern has a potentially large 2D array of pattern instances associated with it, which need updating. This would probably demand an unreasonably large amount of computing time and storage capacity. A solution is required.

A simple solution, which I suggest is used initially, is as follows:

A pattern should be more desirable if it has a low probability of generating 1 values; that is to say, if, when used with the training set of images, the pattern tends, during pattern logic application, to generate 0 most of the time and 1 only occasionally. This desirability may be reduced for the first patterns to be added to the system. (This also applies to meta-patterns: a meta-pattern should be more desirable if the pattern instances of the set of patterns that it implies, as a whole, generate 1 infrequently.) This would be a criterion of the process used to assess any randomly generated pattern before it is semi-permanently added to the hierarchy: the effects of the pattern on the system's predictions of low-level pattern instances, when used with the training set, would be more important. A pattern

which is generated early, or which produces particularly good results may be added to the library even if it has a high frequency of 1 pattern instances: the issue is whether or not it is worth it.

When probabilities are assigned to pattern instances at (X,Y) positions, if the probability is below a minimum value it should be treated as if it is zero. As most patterns will generate 1 infrequently, this means that, assuming that probability values are correctly assigned, most pattern instances at (X,Y) positions will be treated as if they have zero probabilities.

The data structure used to store pattern instances at all the (X,Y) positions for a pattern should be a dynamic structure, rather than a simple array. If most pattern instances for a high-level pattern are 0, it is wasteful to use an array to store these. Instead, only pattern instances at (X,Y) positions with probabilities high enough to merit consideration, which are not being treated as if they have zero probability, are explicitly stored. This reduces the data storage requirements and the resources needed to process the pattern instance data. A similar idea is used during pattern logic application: pattern instances which have not yet been assigned a value are not stored and only pattern instances which are assigned values of 1 are explicitly stored in the data structure. A Pascal example of such a data structure is in Appendix 1. There will be some subtleties to the programming of this. A pattern instance could have a low probability that causes it not to be explicitly represented in the data structure, yet still act as an input to another pattern which causes it to have a high probability pattern instance. Appropriate programming will be able to exploit a low frequency of high probability pattern instances, however.

(Although I have stated that, in general, patterns may be selected that produce pattern instances of 1 with low frequency, 1 is an arbitrary value and it could just as easily be 0.)

The idea of all this is that only a small amount of the data that could be stored and processed by the system is explicitly stored and processed.

Some patterns (or, more likely, sets of patterns produced by meta-patterns), may cause such a huge number of pattern instances to be explicitly stored and computed that it is impractical to deal with them all even during assessment of the pattern. This issue may be dealt with by interrupting computation of the hierarchy and ending assessment of a pattern (or meta-pattern) if the number of explicitly computed pattern instances exceeds some upper limit, before generation of any further pattern instances.

This might seem simplistic, and probably is: it is just a starting point. Some patterns might need to be very common. However, this should mainly apply to patterns performing basic functions that are generated fairly early on and may be permitted for them: it cannot be permitted for an appreciable number of patterns to have every pattern instance computed as the hierarchy would become incomputable. A more sophisticated approach might be to assess every pattern according to its benefits, in terms of improving performance in computation of bottom-level probabilities, weighed against the costs, to give a single value indicating the desirability for the pattern. The costs would consist of

the expected processing and storage costs in computing and storing the pattern instance values and probabilities for the pattern and the overhead in merely having the pattern in the library. Once a pattern has a desirability value obtained like this it might be compared against previous desirability values in an approach similar to what has already been described, the pattern being selected if it appears the most desirable of the previous n patterns. Assessing desirability of a pattern like this would automatically create a tendency towards patterns which result in few pattern instances with 1 values. A pattern which caused generation of many pattern instances which needed to be explicitly stored, with no compensating increase in system performance, would appear undesirable. On the other hand, a pattern which causes significant numbers of pattern instances to be explicitly computed, but can pay for this by delivering a large increase in system performance, may seem desirable. The overall tendency, however, will be towards patterns that have few explicitly computed pattern instances. If an approach like this is used, a meta-pattern would be assessed in the same way, with desirability being computed by treating all of its patterns as a single set with regard to benefits and costs. A method like this might be modified to deal with the issue, mentioned above, of patterns (or, more likely, meta-patterns) that cause explicit computation and storage of such an impractically large number of pattern instances that they compromise the process of even assessing patterns (or meta-patterns). If many explicitly computed pattern instances are involved, it could be that a proportion of these are first computed and the benefits against costs examined to see if there is any chance that the pattern (or meta-pattern) is viable before explicitly computing any more pattern instances. It may be that this is needlessly complex, however, and that the simpler idea of an upper limit on pattern instance numbers associated with a single pattern or meta-pattern, already discussed, would be better.

Some patterns may be more useful as inputs to later patterns than they are in their own right. This may be recognized in such an approach. A pattern might be placed in the library if its performance at least meets some lower standard, but removed later if the inclusion of that pattern in the library provides insufficient opportunities to make useful patterns that use its pattern instances as inputs. This could not be done for all patterns, or nothing could ever be decided, but it may be done for some patterns that provide some immediate benefit. Removing a pattern from the hierarchy would require removal of any patterns that use its pattern instances as inputs.

Some patterns that we encounter in everyday life are so general that they seem to be everywhere. Obvious examples would be things like “squares”, “circles”, or the idea that an object may be part of a pair if other pairs are seen nearby. Patterns like this might, because of their generality, be expressed as meta-patterns. It may seem that pattern instances like this should be extremely common in the hierarchy, but intuition can be misleading on this. Being common in our experience does not translate to being common in the hierarchy. Most of the hierarchy should contain trivial pattern instances that are not explicitly represented, due to low pattern instance probabilities, and even patterns associated with very general meta-patterns would not be expected to generate many explicitly stored pattern instances from these. In human terms, parallel lines may seem to be very common to us – in building walls, on desks, in wallpaper patterns, etc – but we

do not usually think about the large numbers of parallel lines that are not there, because to be there *they would have to involve objects that do not exist*. Nor do we think about the non-existent patterns based on such patterns and so on. We might imagine a model fire engine in front of you right now, at some particular coordinates, and other fire engines, at different coordinates. If we regard each model fire engine as having something like a pattern instance probability, then that probability should be so low as to cause it not to be explicitly stored. (I am assuming you are not reading this while there really are model fire engines right in front of you. If there are, please substitute something else that is not in front of you or move somewhere else.) The same can be said about pattern instances derived from these non-existent model fire engine pattern instances, such as the pattern instance of one of these fire engines putting out a pretend fire: if the probabilities of model fire engine pattern instances are so low that they do not get explicitly represented then neither do any pattern instances based on model fire engines, or pattern instances resulting from them: when a group of pattern instances becomes so low-probability that none of them are explicitly represented then everything dependent on them becomes the same. The same applies to any *abstract* patterns derived from these model fire engine patterns: arrangements of these non-existent model fire engines, such as parallel lines or circles of model fire engines, do not get explicitly stored, or even considered, either. No matter how many parallel lines or circles you think you see, there are many more parallel lines that are not there – those based on arrangements of non-existent things – and what I have said about trying to select patterns that generate 1 infrequently needs to be understood in this context. This example might show how rare the pattern instances of even some of the more general patterns might be in the hierarchy. It should also make it obvious that the frequency with which a pattern instance is explicitly represented is not dependent on how many 1 results it has in its own truth table (or equivalent) so much as on the pattern instances that are providing its inputs.

This is not just intended as a minor improvement, to make things run a bit faster. *With a system which is representing patterns, based on patterns, based on patterns, etc, it is critical that parts of the hierarchy at higher levels are only explicitly represented when there is a reasonable chance that they are based on meaningful information at lower levels*. Failing to use an approach like this, or something similar, will increase the amount of processing by *many orders of magnitude* and it will be impractical to do it.

(An approach like this, intended to counter the situation described in the *Antigonish* poem, might be considered an anti-Antigonish approach – which is presumably a *gonish* approach.)

Further refinements could be made later to make processing of particular pattern instances, or groups of them, conditional on relevance to the computation of probabilities for particular bottom-level pattern instances. It might also be the case that the threshold probability value at which the probability for a pattern instance is regarded as zero is set to different values for different patterns. The system has to be able to work to some degree without these kinds of sophisticated improvements, however, because the basic system establishes the “baseline” with which everything else works: it must be viable in its own right.

A more general version of this sort of approach might be imagined, which, somehow, does not just prevent individual pattern instances from being explicitly computed, but prevents the pattern instances of entire patterns, or groups of patterns, being computed for meta-patterns that imply many conventional patterns. I would suggest that it is not necessary to get into this at this stage.

9 Simple View of the System

Small elements of the image are parts of objects. Abstract patterns at high levels of the hierarchy are merely higher-level objects. Parts of the image statistically suggest the presence of higher-level objects, which statistically suggest the presence of still higher-level objects and so on. Higher-level objects, in turn, suggest the presence of lower-level objects, so parts of an image suggest the existence of multiple levels of higher-level objects, each in turn being used to fill in details in the level below it, ultimately causing image details to be filled in. Higher-level objects are just intermediate results in processing that predicts the existence of lower-level objects from other lower-level objects.

10 Use in General AI

10.1 A General AI System

The approach described here is proposed for use in analyzing images and allowing a computer to determine what missing parts of images are like. This is not its ultimate purpose, which is to provide a modeling system for general AI, to allow a machine to behave appropriately to satisfy various goals.

In earlier articles I discussed the *planning as modeling* approach to AI [1]. In this approach, the problem of planning for a machine is side-stepped by using the modeling system to predict future values of an evaluation function score, similar to the sorts of evaluation function scores used in chess algorithms [9], and based on previous inputs and outputs and previous values of the evaluation function score. This can then be used to determine which outputs to make.

A modeling system like the one described here would be suitable for this use. Changes would be needed to deal with the more general use, but the same general ideas would apply. An AI system built like this would work as follows:

A modeling system like the one described here, but adapted for more general use, has all the inputs and outputs that occur in the AI system fed into it where they become the bottom-level pattern instances. The use of the system is somewhat different now. The bottom-level pattern instances are not the data in an image at some instant in time, but inputs and outputs that have occurred in the AI system over a period of time. A single pattern may use as inputs other pattern instances derived from inputs or outputs which have occurred at different times, so patterns are now *temporal*. A number of different bottom-level patterns may be required for this and the position of an input or output value

in a pattern instance array (or the more sophisticated equivalent of an array which is being used) depends partly on when it occurred. Some pattern instances in the bottom levels for which data is not available correspond to future input and output values. A modeling system like this is therefore making predictions.

An evaluation function continually generates an evaluation function score, based on recent inputs and indicating the desirability of the current situation. Values of this evaluation function score are continually passed to the modeling system, with the inputs and outputs, where they become bottom-level pattern instances.

The modeling system continually analyzes the history of previous inputs, outputs and evaluation function scores. A hierarchy of higher-level pattern probabilities is generated based on these values, and probabilities propagate up it to generate high-level abstractions and downwards to fill in probabilities corresponding to detailed, low-level predictions. At any time the modeling system is therefore predicting future values of the evaluation function. This also means that the system is modeling its own future behavior.

When an output is required, a particular value for that output can be placed in a bottom-level pattern instance of the modeling system as if it had occurred. The modeling system is then made to update itself and adjust probability values in the hierarchy accordingly. The prediction of the future value of the evaluation function score is then obtained by reading the relevant bottom-level pattern instance. This indicates the desirability of making the output with that value. By performing such a process for each of the possible output values, the best output value can be determined.

As discussed in previous articles, *prioritization control outputs* may be used. These would be special outputs made by the AI system directing the modeling system to use particular patterns, or groups of patterns, in its modeling, or possibly to concentrate modeling on particular groups of pattern instances. Previous articles suggested that prioritization control would be used, but I am unsure if they would be needed with this system: there seems to be some scope for preventing unnecessary processing without it and whether or not it is ultimately used depends on how successful such measures are and on what contribution prioritization control can make.

An AI approach like this is based on the idea that planning is based purely on prediction and that prediction is based purely on object recognition. The kinds of objects corresponding to patterns in this kind of use of the modeling system are *temporal objects*, corresponding to events occurring over periods of time. As this article is about modeling I will not discuss the advantages of the planning as modeling approach in detail here. For such a discussion I suggest my previous article on the subject [1].

It is because the modeling system is intended for this use that it has not been designed to provide high-level, abstracted information. For example, the modeling system cannot be directly requested to recognize a specified kind of object, although such a role may be performed by a general planning as modeling system which uses the modeling system and the modeling system internally contains data structures equivalent to high-level

objects. The modeling system merely produces predictions of the low-level data because this is all that is needed for planning as modeling.

10.2 Forgetting

A general AI system as has just been described will be storing a progressively larger amount of historical data corresponding to previous inputs, outputs and evaluation function scores, in the bottom-level pattern instances for the system. This will require progressively greater storage and computing resources. This is impractical. It is unfeasible that human brains work in this way; for example, storing every input through the optic nerve indefinitely would involve a huge number of pattern instances.

The way that this modeling system works suggests an obvious solution, when it is used for general purpose AI. A hierarchy of probability values corresponding to higher-level objects is generated above the inputs, outputs and evaluation function scores on the bottom level, and this means that bottom levels of the hierarchy can be removed while leaving levels above them intact.

Bottom-level pattern instances, corresponding to input and output values and evaluation function scores, would be removed on reaching a certain age. The probability values for the pattern instances above them would remain intact. Eventually, the pattern instances immediately above the bottom level would be removed, but only on reaching a certain age greater than that at which bottom-level pattern instances are removed. The pattern instances in the next level up would be removed on reaching a still greater age, and so on, with progressively greater degrees of abstraction (in higher levels) being removed from the pattern instance data structure as time passes. This would be useless if, after pattern instances were removed from the system, it immediately re-established them by assigning probabilities: removal of pattern instances in this way should both remove the information and prevent the system from attempting to assign probabilities to the corresponding pattern instances. An approach like this should work well in a prototype system. As with the issue of reducing the amount of computing, further progress may be made in this area. For example, it would be better if the decision to retain or dispose of information could be based in some way on the expected relevance of the information to future evaluation function score predictions.

I suggest that this is essentially what happens in human brains when forgetting occurs: we forget details, but retain abstractions of them, eventually forgetting the abstractions and retaining abstractions of the abstractions, and so on. Humans do seem to have a more capable system in which information can be retained based on relevance: otherwise, for example, revising for a test would be useless. Nevertheless, I suggest that the approach I have described, in which progressively higher levels are removed from the hierarchy as time passes, is the basic system in humans, and that it provides a kind of baseline as the default “forgetting” behavior which will occur in the absence of any intervention from more sophisticated systems which may obstruct or demand the forgetting of particular pattern instances or groups of pattern instances. With the example of studying for a test that I just gave, the default behavior of the system would be to discard progressively

higher levels of the hierarchy, but re-reading material might invoke a more sophisticated system that holds some pattern instances back.

For a picture of how this would work, imagine that the levels of the hierarchy are displayed on a computer screen, the bottom level being near the bottom of screen and higher levels being higher on the screen. Any bottom-level pattern instances to the left of the centre correspond to past inputs, outputs or evaluation function scores and any to the right of the centre correspond to ones in the future. As input values, output values or evaluation function scores become available, the corresponding pattern instances are added to the hierarchy in the centre of the screen. The existing information in the hierarchy, at all levels, scrolls to the left to make room, like one of the side-scrolling video games popular in the 1980s, the hierarchy being continually recomputed. At a certain point towards the left of the screen the bottom level of the hierarchy ceases to exist and any bottom-level pattern instances that scroll past this point are erased. The next level up of the hierarchy ceases to exist at a point further to the left, the level above this at a point still further to the left, and so on. (See Figure 2)

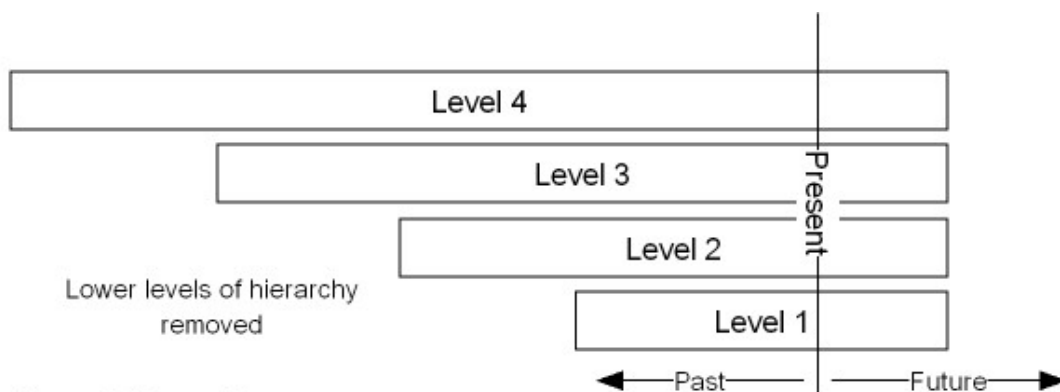


Figure 2: Forgetting

If an approach like this is used, it may be necessary to “fix” the lowest level of pattern instances that are left when those in the levels beneath are removed immediately below them, so that statistics application cannot alter them. This is because the pattern level instances beneath, which were known with certainty, are no longer available and repeated statistics application could cause the system to randomly drift. This would mean treating the level above a level that is removed as if it were the new bottom level.

10.3 Comparison with the Hawkins and Blakeslee System

This kind of system has some similarity with that proposed by Hawkins and Blakeslee [10,11], although there are some differences.

10.4 Comparison with Previous Modeling System

Previous articles [2,3,4,5] proposed a modeling approach with some similarity to this. The previous system represented an attempt at a more general ontology. The equivalent of a pattern was a partial model, a small computer program, and the hierarchy would be constructed by large numbers of such programs. I have adopted the current, simpler approach, instead, partly because the system has to be practically workable when patterns are randomly generated. The previous method of using Turing machines where patterns are now used would deliver little in terms of generality, as truth tables could achieve the same thing on a small scale. If more generality were needed this would be best attained by focusing on how the definitions of sets of pattern instances, sharing common statistics, are represented – and the Turing machines were not used in any way that would help with this anyway.

The previous modeling system used ideas about Occam's razor, rather than the frequency analysis process proposed here. The probability of a partial model taking a particular value was based on knowledge of some of the inputs that it could read and a consideration of the internal workings of partial model algorithm itself, taking its length into account. This process could be computationally inefficient. The approach in this system, however, is intended to produce equivalent results.

The addition of meta-patterns is an attempt to increase the generality of the ontology beyond that of the previous system by allowing the modeling system's experience of the same kind of pattern occurring in different contexts or at different levels of the hierarchy to be expressed.

11 A More General Ontology?

The most important thing about the modeling system is that it is general enough to represent reality adequately. The modeling system described here is not as general as it might be and is actually a special case of a more general system that we might imagine. Patterns describe sets of pattern instances and meta-patterns describe sets of patterns, so the system is based on sets of pattern instances as the "engines" that "see things". Sets of pattern instances are defined in terms of common properties regarding offsets of pattern inputs and possibly the patterns used in them, but a set of pattern instances could be defined more generally. A set of pattern instances could be described by an algorithm, in some general computing language, which somehow specifies a group of pattern instances: a pattern might be replaced by a Turing machine.

An approach like this would make the system more general, but would also have complications. It might mean that some alternative to the arrays (or dynamic structures equivalent to arrays) and coordinates is needed. We might also imagine a system in which pattern instances are statistically related to ones which are somehow in "the same kind of situation" with regard to what they are doing and the pattern instances from which they accept inputs. In all this, we would still need to ensure that most pattern instances are not explicitly computed.

Because systems like this would be significantly harder to design and program, I suggest they do not need trying until the capabilities of a system like the one I describe here are known. If a more general approach like this worked then we should expect the approach described in this article to work to some extent and we cannot be sure that we need to pursue the greatest amount of generality.

12 Conclusion

A probabilistic, hierarchical modeling system has been proposed, based on patterns. Statistical analysis of hierarchies constructed from known data is used to analyze the frequency with which various combinations of inputs occur, this information being stored with each pattern. Probability values influence those above them in the hierarchy, which in turn influence those beneath.

Meta-patterns have been suggested. A meta-pattern defines a group of related patterns and expresses the reoccurrence of a type of pattern in different contexts or at different levels of the hierarchy.

Computing and data storage requirements could be reduced by favoring patterns that provide a 1 value infrequently and not storing information in the hierarchy for cases where the output is 0 or where there is a low probability of it being 1. This idea could be used in a method that determines the desirability of each pattern according to the benefits that it provides in improving the system's ability to compute low-level probabilities against the costs of having the pattern in the library and computing pattern instances for it: an approach like this would naturally favor patterns which infrequently generate 1 values. More sophisticated methods might take account of the contribution of various parts of the hierarchy to the system's predictions.

The proposed method involves continually deriving higher-level object probabilities from low-level data, but keeping low-level data permanently would be expensive in terms of storage and processing needs. A simple erasure method could be used in which information in lower levels of the hierarchy is removed when it has been kept for a certain amount of time, information at higher levels being removed when it has been kept for still longer. Forgetting in humans may use similar principles. More sophisticated methods might be developed which take account of the relevance of information.

An important issue with this system is whether or not the ontology is general enough to represent reality. It may be that it needs modifying to make it more general. It may be that something better than the "pattern instance", as I have described it, is available as the basic computational unit of the system. It may be that it would be better for the system to be somehow less organized with regard to dimensions and coordinates than this one: these sorts of ideas could be employed in a "messier" system. I suggest, however, that the general idea would still be as described here and that this system could have uses.

One way that the ontology might be made more general is by going beyond the simple offset system used here. Pattern instances are the basic computational unit. *Sets* of pattern instances are described by patterns and sets of patterns are described by meta-patterns. This is all about defining sets of pattern instances which can have statistics derived for them and used to assign probabilities for other pattern instances in the set with partial knowledge of their inputs. It could be necessary for patterns and meta-patterns to be able to describe sets of pattern instances more generally than can be done just by using offsets. For example, a pattern could be an algorithm that describes a set of pattern instances which are related in some more complex way than simply sharing offsets. I have avoided this for now to try to get something that is practically feasible, given that the patterns and meta-patterns have to be randomly generated, but it could be investigated further.

I suggest that the modeling system described here is an approximation of how modeling works in human brains, though without any one-to-one correspondence between pattern instances and neurons. It might be interesting to consider how the system may be compromised if modified in some ways, for example by allowing patterns to have more inputs, or allowing patterns or meta-patterns to cause explicit computation of more pattern instances and still be desirable, and how this might relate to issues in human neurology.

13 References

- [1] Web Reference: Almond, P. (2007). *Planning As Modelling in AI: A New Version*. Retrieved 29 July 2007 from <http://www.paul-almond.com/PlanningAsModellingNew.pdf>.
- [2] 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>.
- [3] 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>.
- [4] Web Reference: Almond, P. (2006). *Occam's Razor Part 8: Modelling in Artificial Intelligence*. Retrieved 9 June 2006 from <http://www.paul-almond.com/OccamsRazorPart08.pdf>.
- [5] Web Reference: Almond, P. (2006). *Downward Transfer of Probabilities in AI*. Retrieved 15 October 2006 from <http://www.paul-almond.com/DownwardTransferOfProbabilitiesInAI.pdf>.
- [6] Gurney, K. (2002). *An Introduction to Neural Networks*. London: Routledge. Chapter 4, pp49-50. (Originally Published: 1997. London: UCL Press).
- [7] Ibid. Chapter 2, pp13-15

[8] Rietman, E. (1988). *Experiments in Artificial Neural Networks*. Pasadena: Tab Books. Chapter 2, pp21-23.

[9] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 2, pp7-37.

[10] Hawkins, J., Blakeslee, S. (2004). *On Intelligence*. New York: Henry Holt.

[11] 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>.

14 Note

I have used US English spelling conventions in this article, which is different to my previous practice. This is due to the relative numbers of people likely to be reading it in different geographical regions.

Appendix 1: Example Pattern Instance Data Structure

This example is in standard Pascal and shows how an array of pattern instances for a single pattern can be stored economically if only a small number of the pattern instances are going to be explicitly stored.

```
type
```

```
TBinary = (0,1,Unassigned);  
{This is only relevant to pattern instances generated by pattern logic  
from known data, about which there is certainty. All other pattern  
instances have only probabilistic values.}
```

```
TPatternInstancePointer = ^TPatternInstanceRecord;
```

```
TPatternInstanceArray = array [1..2,1..2] of TPatternInstancePointer;  
{The structure is based on a 2x2 array. Each element of the array has a  
pointer to a further, similar, 2x2 array. Records at the bottom of the  
structure refer to individual pattern instances. Obtaining a single  
pattern instance involves following a trail of pointers to the bottom  
of the structure, each specifying the location of the pattern instance  
with greater accuracy - or until a nil pointer is encountered,  
indicating that there are no pattern instances beyond that pointer.  
Selection of each pointer divides the size of the region where the  
pattern instance of interest is (in terms of numbers of pattern  
instances) by 4, so the trail of pointers would be short.}
```

```
TPatternInstanceRecord = record  
    case Terminal:boolean of  
        True: (Probability:real; Result:TBinary);  
        False: (NextValues:TPatternInstanceArray)  
    end; {end of TPatternInstanceRecord record}
```

```
{The Terminal field is True if this record is at the bottom of the  
structure and corresponds to an individual pattern instance. Otherwise  
it has a further 2x2 array of pointers to allow more accurate  
specification of where the required pattern instance is. When the  
record describes an individual pattern instance record, two fields are  
used: a Probability field for use in statistics application and a  
Result field for use during pattern logic application and when the  
inputs for a given pattern are known. It would be possible to do  
without the Result field and use Probabilities of 0 and 1 during  
pattern logic application.}
```