Physica D 42 (1990) 188–201
North-Holland

CONCERNING THE EMERGENCE OF TAG-MEDIATED LOOKAHEAD IN CLASSIFIER SYSTEMS

John H. HOLLAND
Computer Science and Engineering, Psychology, The University of Michigan, Ann Arbor, MI 48109, USA

This paper, after a general introduction to the area, discusses the architecture and learning algorithms that permit automatic parallel, distributed lookahead to emerge in classifier systems. Simple additions to a "standard" classifier system suffice, principally a new register called the *virtual strength register*, and a provision to use the bucket brigade credit assignment algorithm in "virtual" mode to modify values in this register. With these additions, current actions are decided on the basis of the expected values associated with the "lookahead cones" of possible alternatives.

1. Introduction

Whenever one studies adaptation or machine learning in realistic contexts one constraint soon comes to occupy a central position: Feedback about performance is intermittent and lacks detail. Samuel [1], at the very start of the modern study of emergent computation, realized that games – checkers is the example he used – provide a good paradigmatic example of the problem. During the play of a game there is a great flow of information, but only at the end of the game is there any feedback about performance, the game's payoff, and that is only a few bits of information. In more complex environments, such as ecological, economic or social settings, long sequences of actions are typically required before some reward or reinforcement occurs (e.g. reduction of a "drive" like hunger, or the filling of some "reservoir"). Somehow the system must utilize the "sparse" information about performance, and the large flow of other kinds of information, to improve its performance over time.

Samuel offered lookahead as a (perhaps, the) solution to this problem, and provided a convincing demonstration of its efficacy. To utilize lookahead a system must generate an internal model of its environment. This model enables the system to "look ahead", allowing it to make predictions about the expected consequences of different sequences of action. These predictions can be checked as experience accumulates, providing feedback that can be used directly in improving the model. Note that the resulting procedure for modifying the emergent model depends not at all on environmental measures of performance. Of course, if the model is to be the basis for improved performance, some of the predictions at least must be concerned with expected rewards or reinforcements. However, model-based lookahead neatly steps around a requirement for continual detailed information about performance, and it makes good use of the large flow of (non-performance) information supplied by the environment.

This paper is concerned with the emergence, in classifier systems, of organized, rule-based models that permit "lookahead". (The paper is a continuation and elaboration of the paper on classifier systems that appeared in 1986 in Physica D 22 [2]. Section 1 of that paper details the definitions and notations used here; however, most of the relevant ideas are sketched below in figs. 1, 2, 5 and 6.)

2. Symbols and internal models

An internal model is above all a "symbolic" entity and its predictions depend upon an appropriate manipulation of those symbols. Most current computation-based approaches to symbols and symbol-processing can be assigned to one of two broad classes: The "language-based" systems, such as those implementing the physical symbol system hypothesis [3], and the "stimulus-oriented" systems, such as those investigated by the connectionists (see, for example, ref. [4]). In general, physical symbol systems are good at lookahead, anticipation, and means-ends analysis, when supplied with an appropriate internal model of the environment (usually given as a problem space), but they generally lack procedures for the autonomous construction of experience-based internal models (new problem spaces). Connectionist systems are good at the autonomous construction of categories on the basis of knowledge acquired while exploring an environment, but they lack procedures for organizing that knowledge into models that guide the system by lookahead and anticipation.

It is important that both kinds of system, different as they are in most respects, share a common characteristic: Information about the environment, as supplied by the input interface, always comes with "labels" of some kind. These labels may be quite sophisticated (such as labeling a given input image a "chair") or they may be quite primitive (such as the retinal coordinates of an input neuron). The question, in generating internal models, is not whether or not input is labeled, but rather how sophisticated the labels are. Stated another way, it is a question of how much "intelligence" the input interface uses in translating the environment into the input messages processed by the system.

Both kinds of system thus share a common limitation on their ability to categorize the external world: Environmental states that cause the input interface to generate the same input "message" are indistinguishable, and further processing, however implemented, can only categorize the distinguishable. Indeed, this limitation is shared by any system that acquires all its environmental information via an input interface. If such a system is computationally complete with respect to sorting input "messages" into categories, then it has reached the limits of what categorization can do for it.

Clearly, when it comes to building models, there is a great difference between a system that has a selection of higher-level categories "wired" into its input interface and a system that uses only primitive, coordinate-like labels to formulate higher-level categories. In the latter case, categories and symbols, assuming they emerge, tend to be constructed of "building blocks" – new categories and symbols are constructed by using "good" building blocks, and experience is thereby transferred to new situations. In the former case, categories and symbols tend to be monolithic and experience must be transferred from one domain to another by other means.

Taking this into account, there are reasons that both the stimulus-oriented and language-based approaches should pay close attention to Edelman's [5] points about "re-entrant connections": A system can only generate autonomous internal activity – and lookahead is an example par excellance of such an activity – if it has "re-entrant connections". This point is closely allied to the one Hebb [6] makes in his magnum opus *The Organization of Behavior*: Re-entrant connections provide a recirculation of pulses that allows parts of the network to act independently of recent inputs. As the system learns, clusters of neurons use some of the re-entrant connections to form reverberating "cell assemblies", and these in turn become building blocks (a kind of flexible "compositionality", à la Pylyshyn [7]) for sophisticated sub-routines called "phase sequences". Several nerve net simulations of the 50's, now largely forgotten (e.g. Rochester et al. [8]), exhibited the emergence of "cell assemblies", under Hebb's learning rule, when repeating input patterns were applied to randomly connected networks with re-entrant connections.

For the stimulus-oriented connectionists this point bears on the construction of internal models in another way: It is a long-established theorem of automata theory (going back to McCulloch and Pitts [9]) that a system constructed of interconnected "logical" elements (such as formal neurons), without internal feedback loops, can attain only a very limited subclass of the class of finite automaton behaviors. For example, networks without internal feedback loops cannot exhibit indefinite memory ("at some time in the indefinite past, event x occurred"). Accordingly, without such loops, it is impossible to construct an internal storage for pulses or a counter for pulses. It follows that most computational routines are impossible for nets without loops. In particular, internal feedbacks are necessary if the networks are to be able to produce emergent, semi-autonomous internal models that provide predictions and anticipations.

At the other end of the scale, language-based systems are almost always computationally complete because they directly employ some "universal" language such as LISP. However, they have little to say about the emergence of categories and internal models under the impetus of experience. This is partly the result of using symbols that are pre-defined and close to monolithic, and partly the result of designing systems that require inputs ("symbols") that activate appropriate sections of a high-level interpreter. It is difficult to design physical symbol systems that can learn using the "low-level" data supplied by natural environments. The learning mechanisms used for language-based systems (such as the "chunking" mechanism used by Laird et al. [3] in Soar) look much more like compilation than like the origination of new categories.

Classifier systems occupy a middle ground between these two approaches. They construct models by using experience to extract simple substructures (building blocks) that can be combined in a variety of ways to yield plausible models. (Hebb makes allowance for similar possibilities by providing for the recombination of parts of cell assemblies via processes he calls "fractionation" and "recruitment".) It is important that these building blocks must be used in a fluid, context-dependent way. If we think of the resulting models as complexes of symbols, then, in the sense so well described by Hofstadter (in chapters XI and XII of ref. [10]), the symbols must be *active*. In a later discussion on the topic of active symbols [11], Hofstadter quotes E.O. Wilson:

"'Mass communication is defined as the transfer, among groups, of information that a single individual could not pass to another.'"

and then goes on to say:

"One has to imagine teams of ants [read "neural firings"] cooperating on tasks, and information passing from team to team that no ant ["neuron"] is aware of...

...[W]hat guarantee is there that we can skim off the full fluidity of the top-level activity of a brain and encapsulate it – without any lower substrate – in the form of some computational rules. To ask an analogous question, what guarantee is there that there are rules at the "cloud level" (more properly speaking, the level of cold fronts, isobars, trade winds, and so on) that will allow you to say accurately how the atmosphere is going to behave on a large scale?...

The difference between my active symbols ("teams") and the passive symbols (ants, tokens) of the information-processing school of AI is that the active symbols flow and act on their own. In other words, there is no higher-level agent (read "program") that reaches down and shoves them around."

It is the notion that symbols are composed of building blocks that can be recombined fluidly in response to context that makes emergence of symbols and internal models a natural, almost inevitable, process.

Different combinations of building blocks yield different internal models that compete and gain varying degrees of confirmation as experience accumulates. Parts of highly confirmed models, and

sometimes whole models, serve as building blocks for still more sophisticated models and competitions. (As one implementation of this notion, see the parallel, rule-based, message-passing systems discussed in Holland et al. [12].) In principle, such a system could yield an "upper" layer that behaves much as described by the physical symbol system hypothesis. However, when it comes to the origination of new hypotheses and models, the upper layer is the servant of the lower layers. Whether one prefers the stimulus-oriented or the language-based approach, it seems to me a great risk to ignore processes that construct models by extracting and combining building blocks.

3. Classifier systems and marker-passing lookahead

The remainder of this paper is devoted to an outline of a classifier system (a distributed, parallel system, hence essentially connectionist) that is rule-based and lookahead-oriented (hence akin to physical symbol systems that use means–ends analysis). It continually augments its models by adding rules and proto-symbols (tags) as it gains experience in its environment. The objectives of the design are to (i) use a small set of domain-independent "local" mechanisms to (ii) provide for the emergence of a hierarchical, epoch-guided lookahead based on experience.

A standard classifier system involves a set of message-passing rules in condition/action form. The action part of a rule specifies a message that is to be posted when it is executed. A rule is only executed when there are messages present that satisfy its condition part. Many rules can be active simultaneously. Overt actions (affecting the environment) are the result of messages directed to the output (effector) interface. (See figs. 1, 2 and 5, and, for more detail, Holland [2].)

Lookahead amounts to an extension of this system wherein the system attempts to predict the effect of a sequence of actions so as to base its current overt action on expected long-term consequences. (For example, in the game of checkers, the program decides to move a checker to the edge

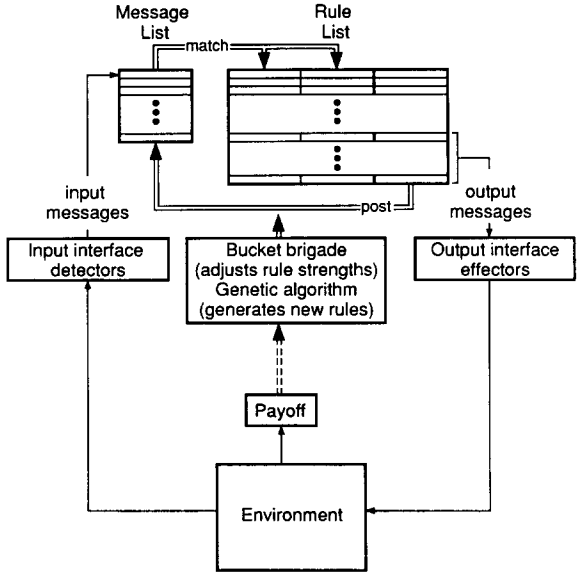

Fig. 1. General organization of a classifier system. Each *rule* is in condition/action form and has an assigned strength that reflects its past usefulness. On each time step, all conditions are checked against the *message list* for matching messages. If all the conditions of a rule are matched then it competes in terms of its strength to post the message specified by its action part. Many rules may win the competition, with many messages being posted for processing on the next round. The *input interface* provides one or more messages that describe the current state of the environment, and the *output* interface translates some messages into actions that affect that environment. Learning takes place by (i) modifying the strengths of rules to reflect experienced usefulness, and by (ii) generating new rules to replace weak rules. The overall performance of the system is measured in terms of *payoff* it receives from the environment.

of the board, anticipating that the move will make possible a "double-jump" four moves hence.)

Because classifier systems are parallel systems it is natural to think of a lookahead process that traces many possible courses of action simultaneously. *Marker propagation* over a network provides a useful metaphor for exploring this possibility. (Fahlman's [13] treatise provides a detailed description of parallel marker propagation.) In the present context we can describe the network, and marker propagation over it, as follows (see figs. 3 and 4):

(1) Each node in the marker propagation network corresponds to some equivalence class (category) over the environmental states. The directed

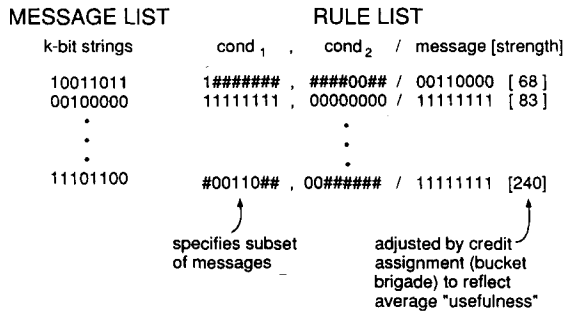

Fig. 2. Specification of messages and conditions in a classifier system. "#"'s in the condition part of a rule act as "wild cards" or "don't cares", accepting any value at that position in a message. A condition with more #'s is more *general* in the sense that it accepts a broader range of messages. Rules with matched conditions *bid* to post the message specified by their action part. The bid is equal to c(rule strength)(rule specificity) where c is a fraction (say $c = 0.1$) and rule specificity is given by k − (no. of #'s). Winners are chosen with a probability proportional to the size of their bids.

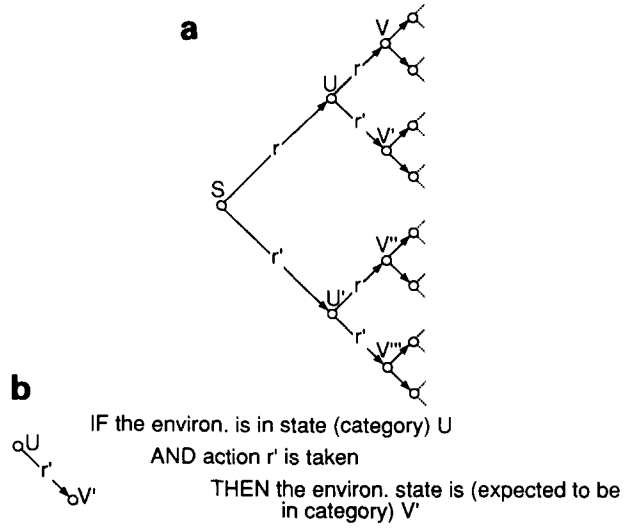

Fig. 3. An example of a causal network. (a) Fragment of a "causal net" version of an internal model. (b) Interpretation of an edge of the causal net in terms of rules.

edges connecting the nodes correspond to possible actions that will cause state transitions in the environment. (The directed edges amount to hypothesized causal relations.)

(2) The lookahead process is initiated by placing a marker on the node corresponding to the current state (more carefully, the equivalence class containing the current state).

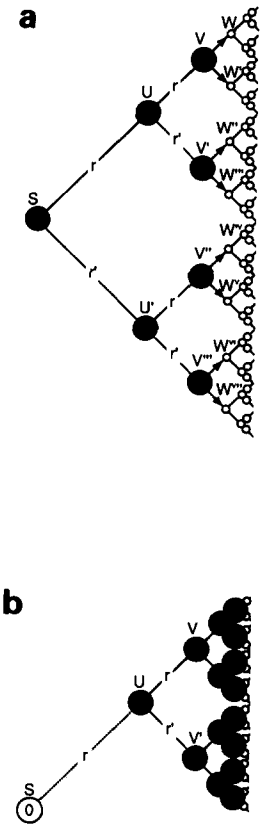

Fig. 4. An example of lookahead using marker propagation. (a) Marker propagation prior to an action decision at state S. (b) Marker propagation after action is carried out at state S.

(3) Copies of the marker are then propagated along each directed edge leading from that node, the result being that the initial node and all nodes that can be reached from it in one step are marked.

(4) The process is iterated T times, with the result that a "cone" of nodes is marked representing the states (equivalence classes) that can be attained from the current state via various action sequences of length T.

This elementary process can be made more sophisticated in several ways. Among the possible extensions, the following two play a key role in the extended classifier system:

(1) The system's experience will typically be indeterministic. That is, a given action applied to a given node (equivalence class) leads to different consequences (equivalence classes) at different

times. (This happens because the equivalence classes are too "coarse" to capture some distinctions necessary for a fully causal description.) Under such circumstances the edges can be assigned probabilities corresponding to the relative frequency with which the alternatives occur. Marker propagation still proceeds in parallel, but probabilistically. The lookahead "cone" now contains action sequences of various lengths, lookahead proceeding to greater depth along more probable paths.

(2) Equivalence classes often have "values" attached to them. (The evaluation functions used in game-playing programs provide a concrete example – the evaluation function provides an estimate of the general value of attaining a given equivalence class.) To induce the system to explore more valuable paths, the probability of propagation can be made proportional to the product of the probability assigned to the edge and the value assigned to the node at the tip of the edge. This number is akin to the expected value of the transition. With this addition, the length of each path in the lookahead "cone" varies roughly as the average expected value assigned to that path.

Note that the purpose of lookahead is to influence the decision as to what action is to be taken next. That is, the current action is predicated upon an estimate of the future value of moving in that general "direction". (More carefully, the first step away from the current state is chosen on the basis of the estimated values of the alternatives attainable in the part of the lookahead cone that includes that first step.) Note also that, once that first step has been taken, the parts of the lookahead cone involving other possible first steps are now largely irrelevant. However, the part of the lookahead cone involving the chosen first step applies and can be extended – marker propagation can continue from the *end-points* of the sub-cone, rather than starting all over at the new "current" state.

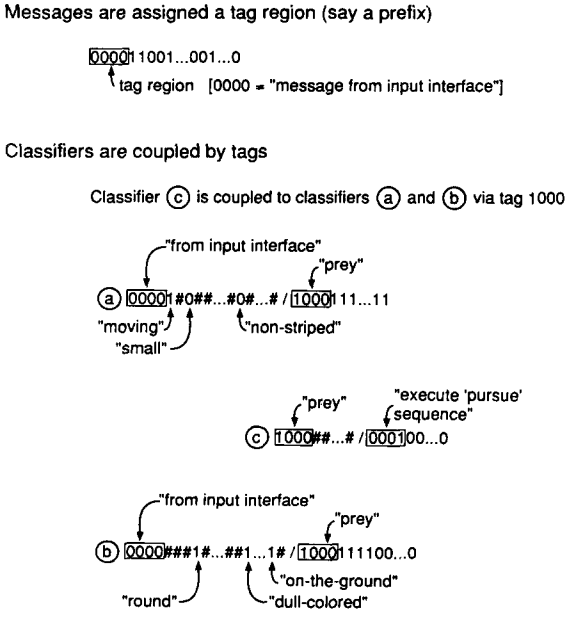

Fig. 5. Tags and rule coupling.

4. Tag-mediated lookahead

The first step in providing a classifier system with lookahead is implementation of the "causal" network that constitutes the system's internal model of its environment. For each transition to be modeled, this is a matter of implementing the rule "IF the environment is in state S AND action A is taken THEN (the system expects) state S′ will occur". This can be accomplished by coupling a rule that is active when state (category) S is detected to a rule that is active when state (category) S′ is detected. In classifier systems, tags (see fig. 5) typically provide the couplings that implement pointers, action sequences, and the like. Thus, tags play a central role in the construction of the causal network, and they constitute natural building blocks for emergent models.

Tags are implemented by setting aside certain regions in internal messages, typically a prefix or a suffix, though any region will do. For example, any message with the prefix 1101 will satisfy a condition of the form 1101# ... #. Stated another way, a classifier with the condition 1101# ... #

has an "address". To send a message to it, simply put the prefix 1101 on the message. There are many variations on this theme. For example, consider a pair of classifiers C1 and C2 that send messages tagged with 1101 and 1001, respectively. A classifier with the condition 1101#...# will attend only to C1, but a classifier with condition 1#01#...# will attend to both C1 and C2. Moreover, a message with a given tag can activate a whole cluster of classifiers, if all the classifiers in the cluster have conditions that are sensitive to that tag. Fig. 5 provides an example of this use of tags, and the interested reader will find a detailed description of the use of tags to implement a semantic net (KL-One) in Stephanie Forrest's paper [14].

Tags supply the "glue" for models in classifier systems and, as with any other part of a classifier, they are subject to modification and elaboration under the recombinations induced by the genetic algorithm. As the genetic algorithm supplies the system with additional coupled rules, the tags acquire meaning in terms of the model-based effects they mediate (actions, anticipations, predictions, and the like). In effect, tags serve as active proto-symbols providing context dependent associations: Many classifiers can be activated by a message with a single tag – the particular cluster activated being dependent on the other messages present.

As the system evolves, it seems reasonable to expect that these proto-symbols will become associated with external, manipulatable patterns (physical symbols). These external patterns, feeding back through the input interface, would "close the loop", moving the proto-symbols to full-fledged symbols with distal access.

5. Hierarchical models

The internal models that arise naturally in the classifier system format are best described as *default hierarchies* (see fig. 6). The (useful) rules that are easiest for the system to discover are those with many #'s (don't cares) in the condition part. This is true both because such rules are easy to formulate and because they are tested often. Any such rule that gives the system a slight statistical advantage will be quickly strengthened. These rules act as *default* rules. More specific rules that contradict the default rules in specific situations are tested and established less quickly. These rules act as *exception* rules. There can of course be excep-

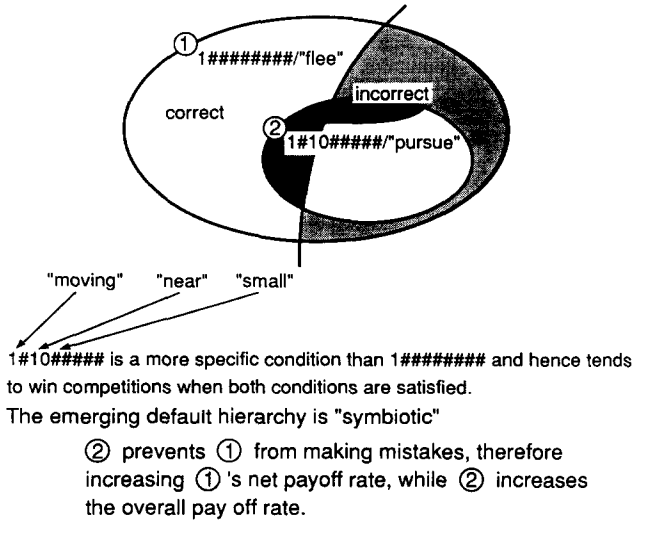

Fig. 6. An example of a simple internal model in a classifier system.

tions to the exceptions, and so on, whence comes the *default hierarchy*. Fig. 6 illustrates the point that, in classifier systems, these contradictory rules, rather than competing to displace each other, can act in a symbiotic fashion. Because of this, default hierarchies form a "natural" emergent structure in classifier systems.

The elements (rules) in the hierarchy can also respond to events of different duration. For example (see fig. 7), there could be a rule that has a condition that is satisfied so long as it is daytime and the "food reservoir" (stomach) is unfilled. Such a rule could continue to post its message over a considerable time interval, which could be called the HUNT epoch. There could also be a more specific rule that has its condition satisfied under the same conditions but only so long as there is no "prey" in sight. This rule would be active for only a part of the HUNT epoch, a sub-epoch that could be called the SEARCH epoch. Again a hierarchy forms, an *epoch* hierarchy, involving conditions that are increasingly specific (as in the earlier default hierarchy) and activities of progressively shorter duration with more detailed specification.

In the figures and discussions that follow, it is useful to keep in mind a simple system–environment configuration, wherein the state of the configuration can be described as a vector over four properties:

$$\{\text{day}, \text{night}\} \times \{\text{hungry}, \text{not hungry}\}$$
$$\times \{\text{prey (in sight)}, \text{no prey (in sight)}\}$$
$$\times \{(\text{prey}) \text{ off-center}, (\text{prey}) \text{ centered}\}.$$

Actions (the edges of the transition graph) can be restricted to the set:

$$\{\text{"run"}, \text{"twiddle"}, \text{center}, \text{forward}\}.$$

The intended action sequences of the system could then be collected into equivalence classes with various levels of refinement and duration (see fig. 7). At the coarsest level, the system's action, "hunting", is an activity that could last the better part of a day and would consist of admixtures of the elementary actions "run", "twiddle", etc. Early in the system's experience the epoch hierarchy would consist of a single level and this admixture of elementary actions would be more or less randomly determined. With experience, the epoch hierarchy supplies additional levels of specificity and the admixture becomes more context dependent. That is, the coarsest equivalence class, which extends over both "space" (different instantaneous states) and "time", is progressively refined into classes of shorter duration and fewer states, yielding an epoch hierarchy.

6. The lookahead process in classifier systems

We are now in a position to look at an outline of the mechanisms necessary for a classifier system to produce emergent experience-based internal models. *The basic objective is to add lookahead to a standard classifier system without adding new rule types or changing the operators that generate the rules*. In particular, this means that the lookahead process should use the same coupled rules that are generated by the "triggering" processes in the standard classifier system (see section 3.2 of ref. [12] or section 9.3 of ref. [15]).

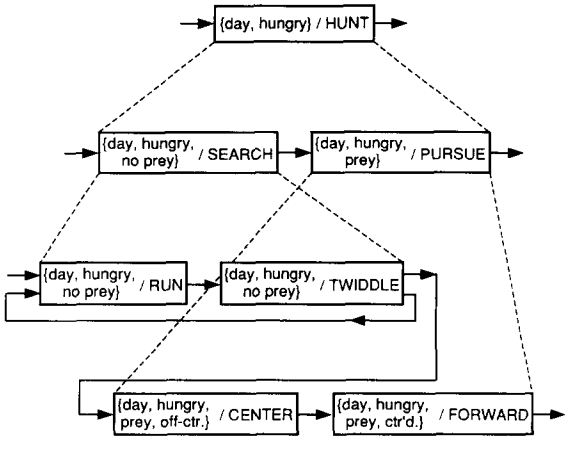

Fig. 7. An example of an epoch-directed hierarchical model (see text).

The main triggering mechanism is that for producing coupled rules (see p. 90 of ref. [12]). The paradigm case occurs when:

(1) Rule R is executed at time t,

(2) Rule R′, not coupled to R, is executed at time $t+1$,

(3) R′ is substantially strengthened (via the bucket brigade credit assignment algorithm or by direct payoff from the environment).

This condition triggers the generation of a pair of coupled offspring rules, R1 and R1′, that are activated under the same conditions as R and R′ but are coupled by a (newly generated) tag. The pair so generated does *not* replace the parents, but simply enters the system as a competing hypothesis. If the coupled pair captures a causal relation in the environment, it will be strengthened under the bucket brigade (because of the profit made by R1′) and it will persist. If not, it will quickly lose strength (because R1 fails to set the stage for the activation of R1′), becoming a candidate for replacement by other newly generated hypotheses.

There are three types of rules that serve as the elements of a causal network in a standard classifier system:

(1) *Node* rules. The condition part of a node rule is satisfied by a message designating a particular state (category). The action part of the rule sends a message indicating that the state has been "marked".

(2) *Transition* rules. The condition part of a transition rule is satisfied by a message designating a particular state (category). The action part of the rule sends a message designating a response (the label of the corresponding edge in the causal network) and the state (category) expected when that response is made to the state designated in the condition part.

(3) *Action* rules. An action rule has two conditions in its condition part: The first condition is satisfied by a message designating a particular state (category), and the second condition requires the presence of a special *action* message. The action part of the rule sends a message to the output interface that causes some overt action in the environment.

The messages produced by these rules have three parts:

(1) a prefix part allocated to tags,

(2) a middle part typically allocated to response specification, and

(3) a final part typically allocated to state specification.

Where helpful in the discussions and figures that follow the three parts will be indicated by a sequence of three pairs of parentheses corresponding to the three parts: ( )( )( ).

Three different tags are used to distinguish different interactions between the coupled rules representing the causal network:

(1) An "i" ("input") tag indicates that the message originates from the input interface.

(2) An "a" ("action") tag indicates that the message is directed to the output interface.

(3) A "v" ("virtual") tag indicates a message that is involved in node marking. There are two subtypes to the v tag:

(i) A "v0" tag is used on messages that initiate the marking process.

(ii) A "v1" tag is used on messages involved in an ongoing marking process.

The lookahead process depends upon the fact that the node and transition rules are coupled so that messages tagged with (v) are propagated just as messages tagged with an (a). However, when the message is tagged with a (v), actions upon the environment are not carried out, and the "next states" are those anticipated by the transition rules under the response (r) specified by the message.

Fig. 8 uses these conventions to show in detail how a classifier system would carry out the marking process illustrated in fig. 4. The caption for the figure gives an overview of the process.

7. The virtual bucket brigade

We emphasized at the start that the purpose of lookahead is to allow a system to make current action decisions on the basis of anticipated future consequences of those decisions. So far we have described a classifier system that can use causal

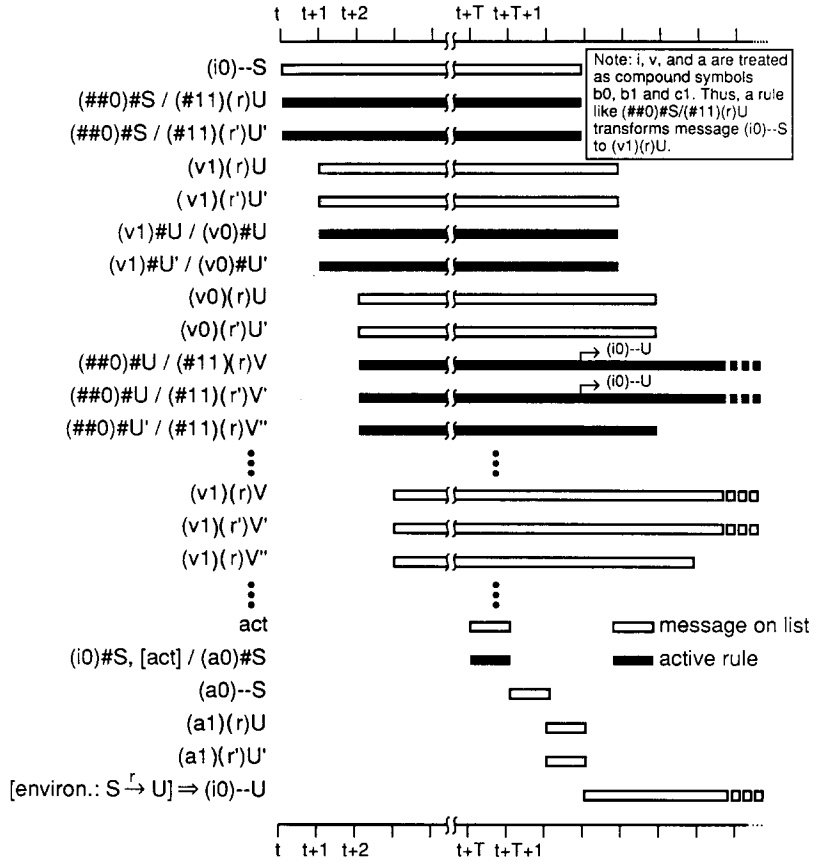

Fig. 8. Timing diagram for classifiers implementing the lookahead process of fig. 4. The process begins, on the top line of the diagram, when a message (i0)--S comes from the input interface, indicating that the environment is in state (category) S. This message starts the lookahead process by activating the transition rules on lines 2 and 3 of the diagram. The "pass through" # in the action part of these rules transforms the (i0) tag to a (v1) tag producing the messages (v1)(r)U and (v1)(r')U', at time $t + 1$ on lines 4 and 5, indicating that markers are passing over the corresponding edges in the causal network. These messages activate the node rules on lines 6 and 7, issuing the messages (v0)(r)U and v(0)(r')U', at time $t + 2$ on lines 8 and 9. These messages initiate further marker passing from the nodes corresponding to U and U'. This basic process is continued until an "act" signal appears at time $t + T + 3$, indicating that the system must take some overt action. This causes the action rule associated with state S, on the fifth line from the bottom of the diagram, to be activated yielding the message (a0)--S. This message now causes the transition rules on lines 2 and 3 to issue messages (a1)(r)U and (a1)(r')U', which, because of their tags, are directed to the output interface. There they compete since they specify different responses. (Lookahead affects this competition through the *virtual bucket brigade* discussed below.) In this example, the response r wins out, causing the state transition from S to U to occur in the environment. The new environmental message (i0)--U keeps the transition rules for V and V' (lines 10 and 11) active, but the transition rules for V'' and V''' (line 12 and ...) go inactive, in effect dropping the markers from the latter nodes because they are not accessible from the new state U.

couplings (acquired through triggering) as the basis for marker-passing lookahead. We have yet to indicate the way in which this lookahead can influence current decisions. Once again Samuel [1] leads the way. In the checkersplayer, he associates a "cone" of future possibilities (cf. step 4 of the description of marker-passing lookahead in section 3) with each currently possible response (move). That is, for each currently possible move, his program looks down all sequences of moves (to some feasible depth) that begin with that move. He then weighs each first move by applying an evaluation function to the far ends (anticipated future states) of the associated cone. In effect, he

"backs up" the values of future possibilities to the current decision point.

The bucket brigade algorithm does a similar kind of "backing up" over coupled classifiers. Classifiers at the end of a chain of coupled classifiers "pass their strength back", via bids, to earlier classifiers in the sequence (see p. 309 of ref. [2], section 3.1.2 of ref. [12], or section 5 of ref. [15]). This occurs only over an overt sequence of actions (and over several trials of the action sequence), but it does suggest that the bucket brigade might be used for a similar purpose during the "virtual" actions.

To this end, we add a *virtual strength register* to each rule in the system, to supplement the *strength register* already possessed by each rule. (This is the only substantial modification we have to make to a standard classifier system to implement the overall lookahead process.) Now, a rule activated by a message with an (a) tag passes its bid back to the *strength register* of the sender, in the usual way, but a rule activated by a message with a (v) tag passes its bid back to the sender's *virtual strength register*. Each time a rule is *first* activated in a marker-passing (v) process, its virtual strength register is set to the value of its strength register. Thereafter, as long as it is involved in the ongoing marker-passing process, the virtual strength is repeatedly modified by the bucket brigade. Moreover, the bids it makes are based on the virtual strength rather than the actual strength. Under this regime, the associated rules in each cone, once active, stay active as long as they are part of the cone (i.e., so long as the corresponding nodes are "marked"). This means that the bucket brigade is iterated over and over again through those rules. In effect, the virtual strengths come to reflect the value that would have been passed back over *many* overt trials of the action sequence (assuming that sequence accurately predicts the state changes in the environment).

To have the lookahead process influence the current action decision, we need only have the bids of the rules sending action messages be influenced by values in their virtual strength registers. As stated earlier, overt acts occur only when an "act" message is posted. Then, each rule at the start of a lookahead cone (1) posts a message with an (a) tag and a mid-part that specifies a response (see fig. 8), and (2) makes a bid that determines how it fares in the competition to control the output interface. When this bid is influenced by the virtual strength register, the competition at the output interface is affected by the cumulative "backing up" of future values under the virtual bucket brigade. (It is probably sensible to let the actual strength also influence the bid; the relative influence of the two strengths could be determined by a "daringness" coefficient, which could be "wired in" or could be set adaptively by other rules.)

Note that, if the competition between rules is probabilistic, based on the relative sizes of their bids, then lookahead proceeds more rapidly along paths wherein the corresponding transition rules make high bids. (Even though the competition is probabilistic, many winners are allowed at any given time, thereby exploiting the inherent parallelism of the system. See (3) in section 8.) Moreover, under this regime, the virtual bucket brigade produces a sophisticated estimate of the future: The value returned to each rule amounts to the expected value of its lookahead cone (the values of the endpoints weighted by the probabilities of reaching them).

8. Further refinements

(1) Because epoch hierarchies arise naturally when the genetic algorithm is applied to classifier systems, it is worth while to try to exploit them under marker-passing lookahead. One way to do this is to modify the triggered coupling. Under the modification, *node* rules would be formed with two conditions in the condition part. One condition would be the same as before, being satisfied by a message designating the state (category) of the corresponding node. The other condition would be satisfied by the message of some *more*

general classifier active at the same time. That is, the second condition would (often) couple the node rule to high level, "epoch-marking" rules.

Fig. 9 sketches the use of rules to implement this kind of epoch hierarchy. *Support* (see sections 2.3.2 and 4.1.5 of ref. [12]) enables rules (and anticipations) belonging to the coarser epochs to influence the more detailed actions at deeper levels of the hierarchy. In other words, the "desirability" of a given epoch, as measured by the strength of the corresponding rule and the virtual strength supplied by lookahead at that level, adds support (as defined in section 2.3.2 of ref. [12]) to various branches at lower levels, influencing lookahead and decisions at that level. The resulting structure can be interpreted as a quasi-homomorphic image (see section 2.1 and appendices 2A and 2B of ref. [12]) of the environmental dynamics.

(2) For rules dealing with coarse equivalence classes, as in the upper levels of an epoch hierarchy, a given response may at different times lead to states in different categories. That is, the model at that level is ambiguous as to outcomes because it inadequately distinguishes external states. As a result, the triggering operators generate more than one rule for a given category–response pair (S, r). Under such conditions, it is important that probabilities be assigned to each transition rule that leads from S and predicts a different outcome under response r. This probability can be treated

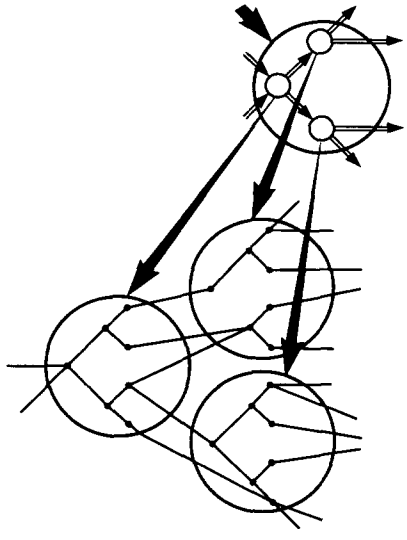

Fig. 9. Sketch of a support-based classifier implementation of an epoch hierarchy.

as a frequency count that is updated each time the response r is overtly executed. The frequency count is increased for the rule R predicting the state category S′ that actually follows the response, and it is decreased for all other rules associated with the pair (S, r). This can all be normalized to a fraction between 0 and 1 by treating the count as an average: Let $P(R, t)$ be the normalized frequency associated with rule R at time t, and let $d(S', t) = 1$ just in case S′ occurs at time t. Then

$$P(R, t+1) = [1 - (1/n)]P(R, t) + d(S', t)/n$$

provides a reasonable updating formula when a constant $n \gg 1$ is used. For example, if $P(R, T) = 0.5$ and $n = 4$, and the corresponding $d(S', t) = 1$ at T, $T+3$, and $T+4$, then $P(R, T+4) = 0.70$, a reasonable approximation to the 0.75 experienced in the last 4 steps. If S″ corresponds to the category S′, and R′ is the transition rule from (S, r) to S″, then $P(R', T+4) = 0.30$, as is appropriate.

Note that, when a prediction is verified, the corresponding rule is also strengthened. Consider, then, two *pairs* of rules:

(i) {(#)(#)(no prey)/(#)("twiddle")(no prey); (#)(#)(no prey)/(#)("twiddle")(prey)}

(ii) {(#)(#)(no prey)/(#)("run")(no prey); (#)(#)(no prey)/(#)("run")(prey)}.

Under either "twiddle" or "run" the result of a search at any instant may be either "no sighting of prey", or "prey comes into view". Past probabilities will determine the relative frequency with which the second rule in each pair wins, which is tantamount to determining the expected length of a "search" before prey is encountered. This, in turn, under the virtual bucket brigade and competition (see (3) below), determines the probability that "twiddle" will be employed over "run".

(3) The list of satisfied rules becomes a sample space, once the associated probabilities of winning are available. This observation provides a useful way for determining the number of rules that will be allowed to win the competition at any given time: The list of satisfied rules is sampled repeatedly until some rule is selected for a second time, at which point the sampling process terminates. The rules so selected post their messages. This process has the advantage that the list will be short if there are a few high-probability rules (the system has well-established means of acting upon the situation), and long otherwise (allowing extensive alternatives, to be resolved in terms of mutual exclusions at the output interface).

(4) In order that competition and limitation of the size of the message list not cause "marked" nodes to be deactivated, the system could be supplied with a special message list for messages from nodes. Note that a special list is only necessary if the node rules are weak. Otherwise the method in (3) provides for an expanding message list with occasional losses – something closer to human lookahead with its difficulty of retaining a conscious picture of all the branches in a "bushy" structure.

9. Commentary

The emergent structure discussed in this paper is an architecture that provides for parallel lookahead under distributed control. Though we have, by now, accumulated considerable experience with learning procedures and emergent structures in "standard" classifier systems (see ref. [15]), we have no experience with classifier systems exhibiting lookahead. For this reason, the new system is organized to exploit structures that are known to emerge under the learning algorithms (the bucket brigade algorithm and triggered genetic algorithms) of the standard systems. It is interesting that a small change in the standard system – addition of a *virtual strength register* coordinated with an additional use of the bucket brigade algorithm in *virtual* mode – provides a sophisticated way for future anticipations to influence current action. In effect, the decisions are based on the "expected" value of the cone of future possibilities associated with each perceived action possibility. It should be emphasized that these ideas have yet to be tested in a complete system – we have experience only with fragments.

Acknowledgements

Many of the ideas presented here have been hammered out over the past couple of years in meetings of the BACH group at the University of Michigan. Dr. Rick Riolo is currently preparing a simulation to test his own version of these ideas in the context of latent learning. The research reported has been supported, in part, by the National Science Foundation under grant IRI-8904203 and its predecessors, and a substantial part of the work was done during visits to the Santa Fe Institute.

References

[1] A.L. Samuel, Some studies in machine learning using the game of checkers, IBM J. Res. Dev. 3 (1959) 210–229.

[2] J.H. Holland, A mathematical framework for studying learning in classifier systems, Physica D 22 (1986) 307–317.

[3] J.E. Laird, P.S. Rosenbloom and A. Newell, Chunking in Soar: The anatomy of a general learning mechanism, Machine Learning 1 (1986) 11–46.

[4] D.E. Rumelhart and J.L. McClelland, eds., Parallel Distributed Processing. (MIT Press, Cambridge, MA, 1986).

[5] G. Edelman, Neural Darwinism: The Theory of Neuronal Group Selection (Basic Books, New York, 1987).

[6] D.O. Hebb, The Organization of Behavior (Wiley, New York, 1949).

[7] Z.W. Pylyshyn, Computation and Cognition (MIT Press, Cambridge, MA, 1986).

[8] N. Rochester, J.H. Holland, L.H. Haibt and W.L. Duda, Tests on a cell assembly theory of the action of the brain, using a large digital computer, IRE Trans. Information Theory IT2 (1956) 80–93.

[9] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.

[10] D.R. Hofstadter, Gödel, Escher, Bach: An Eternal Golden Braid (Basic Books, New York, 1979).

[11] D.R. Hofstadter, Metamagical Themas: Questing For The Essence of Mind and Pattern (Basic Books, New York, 1985).

[12] J.H. Holland, K.J. Holyoak, R.E. Nisbett and P.R. Thagard, Induction: Processes of Inference, Learning, and Discovery (MIT Press, Cambridge, MA, 1986).

[13] S.E. Fahlman, NETL: A System for Representing and Using Real-World Knowledge (MIT Press, Cambridge, MA, 1979).

[14] S. Forrest, Implementing semantic network structures using the classifier system, in: Proceedings of an International Conference on Genetic Algorithms and Their Applications (Erlbaum, London, 1985).

[15] L.B. Booker, D.E. Goldberg and J.H. Holland, Classifier systems and genetic algorithms, Artificial Intelligence 40 (1989) 235–282.