Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

SYSTEM DESIGN AND EVALUATION USING DISCRETE EVENT SIMULATION WITH ARTIFICIAL INTELLIGENCE

John R. Clymer

Applied Research Center for Systems Science
California State University, Fullerton
Fullerton, California 92634 U.S.A.

ABSTRACT

During system design and evaluation, development of optimal decision making rules required to simulate a teleological system can be very difficult. The Operational Evaluation Modeling (OpEM) inductive / adaptive expert system controller, integrated into the OpEM Simulation Tool Kit, can greatly facilitate this task. The decision rule development procedure optimizes system effectiveness by initial induction of the rule tree, using sample decision cases, followed by adaptive rule strength modification to optimize decision making. The rule specification language consists of all primitives needed to implement local and global decision making. These primitives include fuzzy facts to transform analog variables into discrete concepts and variable instantiation of facts to make context sensitive, global decisions. A single-track railroad system is evaluated as an example context sensitive system, and other methods of decision making are compared with OpEM.

1 INTRODUCTION

At certain events during a computer simulation of system operation decisions must be made. Typical decisions determine allocation of resources or choice of alternate actions. In many cases a simple algorithm may make a local, context free decision during a simulation run. As decisions become more context sensitive, the algorithm to evaluate the logic necessarily becomes complex. Such algorithms are difficult to write and modify.

As complexity increases, it becomes attractive to employ a formal system for decision making during system operation. One such system is an 'expert system' that generalizes 'IF' statement logic in a way that can be easily specified and adapted. Input of text data specifies the initial decision making rules, and rule adaptation is performed while the simulation is running.

Use of expert systems in simulations is not new (Buchanan et al 1988, Reddy et al 1986) . However, most applications have been in the area of manufacturing systems, viewing the system as a network of queues and servers. This queuing theory view tends to be context free. The railroad system discussed in this paper is highly context sensitive, making it more difficult to make decisions.

State-event sequences called parallel processes, based on combined discrete and continuous state variables, are used to represent context sensitive system operation (Clymer 1990a, 1990b). For a context sensitive state change, or transition, one event can schedule one or more other events, implementing an event chain. For example, let Di Dj ... Dk be dimensions of the system state space and let discrete events Ea Eb ... Ec indicate value changes in these dimensions. These dimensions represent the functions being performed by system entities or entity state variables such as process counters or motion vectors. A context sensitive transition is described by the rule:

(Di Dj ... Dk) → Ea Eb ... Ec (Dl Dm ... Dn)

This rule indicates that a transition from system state dimensions (Di Dj ... Dk) to values (Dl Dm ... Dn) will occur after event chain Ea Eb ... Ec has been executed. The problem is that there may be thousands of such transitions to choose from when making a decision in a system. It is difficult to know which transition is most effective for each decision case and to devise a set of rules that decides all of these cases correctly. One solution to this problem is to assist analysts in visualizing system operation, so that they can better understand a context sensitive system, and to automate as much as possible the development of optimal decision making rules.

Automating the knowledge engineering process to facilitate decision rule development is highly desirable when modeling context sensitive systems. System engineers can specify the information (facts) needed to

make a decision and they can usually make the correct decision given the facts about a situation. They make these decisions by visualizing desired system operation and specifying the proper decision for each situation. However, systems engineers are usually not good at devising an optimal rule tree needed to make all the correct decisions for a given domain.

The purpose of this paper is to discuss the benefits of using the OpEM inductive / adaptive expert system controller to make decisions that control a simulated, context sensitive system. An evaluation of a single track railroad system is presented as an example of decision making in a context sensitive system. A brief discussion of two of the OpEM expert system controller capabilities is presented: (1) fuzzy facts and adaptive rule strength modification and (2) rule induction from simulation generated cases. Other methods used in simulations for decision making are classifier systems, neural networks, and case-based reasoning systems. These approaches are compared with the OpEM inductive / adaptive expert system controller, and recommendations for future research are made.

2 RAILROAD SYSTEM EVALUATION

2.1 System Scenario

A railroad system consists of two stations (A,E), three substations (B,C,D) between A and E, and one track connecting them, as shown in Figure 1. Two tracks are provided at each substation (B,C,D) and as many as necessary are provided at each station (A,E).

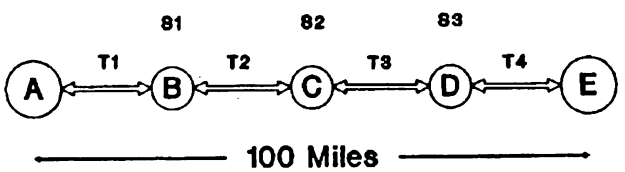

Figure 1: Single Track Railroad Scenario

The track extends beyond stations A and E, but this operation of the system is modeled by the interarrival time of trains arriving at these stations. The track between A and E is 100 miles long. Track segments separating stations, or substations, are not necessarily the same length. The trains move on the track in two directions, either from A to E or E to A. There are two types of trains, fast passenger trains and slow freight trains. A probability is used in the simulation to determine if a train is fast or slow. Trains are scheduled to arrive at station A or E using a Gamma distributed random variable for the interarrival time. The speed of each type of train is also a Gamma distributed random variable.

2.2 Directed Graph Model

The railroad directed graph model is shown in Figure 2. There are five processes: (1) arrival of trains at station A, (2) arrival of trains at station E, (3) train moving from station A to E, (4) train moving from station E to A, and (5) regulator. Processes 3 and 4 are duplicated, providing a separate process for each train. At each wait state WT* (WT1,..,WT4) in a train process, a train waits for its next track T* (T1,...,T4) and for its next substation S* (S1,..,S3). The next track and the next substation must be allocated at the same time to avoid a deadlock. Only one train is allowed on a track at a time to avoid collisions.

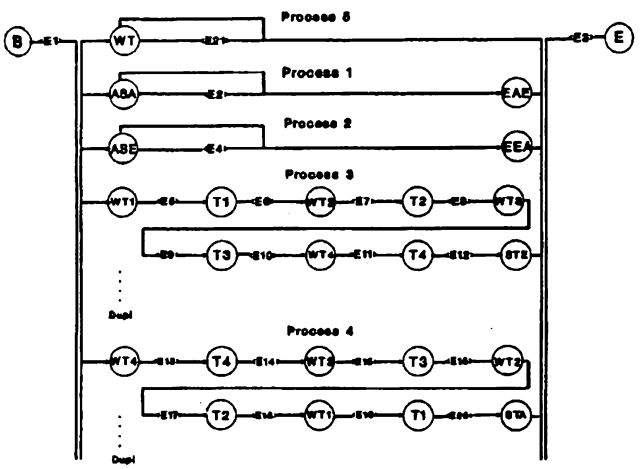

Figure 2: Directed Graph Model of Railroad System Operation

When a train wants to move, it asks the regulator process to make a decision. The regulator process checks the state of each train in the system, the state of each track and parking station, and then decides which trains will be allowed to move, scheduling the next events in system operation. The use of a regulator process is similar to real world operation. Allocation of system resources to a train by a dispatcher who knows the state of all trains and resources in the system is much better than by the computer driving the train that knows only its own state.

2.3 State Space and Knowledge Base

The state variables used in the model to describe the states of system resources are:

SS1, SS2, SS3: total number of tracks available at substations S1, S2, and S3;

S1, S2, S3: number of tracks currently available (not busy) at substation S1, S2, S3; and

T1, T2, T3, T4: number of tracks currently available (not busy) along each segment.

The state of each train, stored in a dynamically allocated moving object record (Corey and Clymer 1991), includes the discrete state, type of train (fast or slow), position, and velocity. In addition to state variables, facts describing the state of system resources and trains are stored in the knowledge-base.

2.4 Situations to be Avoided

Deadlock problem: Figure 3 shows a deadlock that is defined as a situation where no further operation is possible and the mission has not ended. Train 1 and train 3 both require a parking place at substation D and only one parking place is available since train 2 occupies the other place.

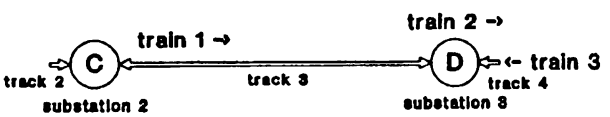

Figure 3: Deadlock Situation

Interferenceproblem: Figure 4 shows an interference that is defined as a situation where more than two trains moving in opposite directions contend for one resource. When train 1 arrives at substation S2, which train should move next. Trains 3 and 4 cannot move until trains 1 and 2 move first to free substation C parking space S2.

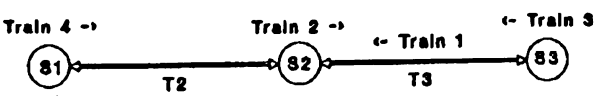

Figure 4: Interference Situation

BlockingProblem: Another kind of problem occurs when different types of trains move in the same direction on the same track. Figure 5 shows a blocking problem that is defined as a situation where a slow train is in front of fast train so that the fast train has to follow the slow train. Fast train 3 follows slow train 1, slowing down the fast train.

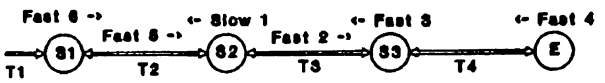

Figure 5: Blocking Situation

2.5 Decision Making Policy

To avoid these problems, a decision making policy was derived to control the trains by making decisions during simulated system operation.

Rule 1: Any train can move if a track to and parking place at the next substation is available; however, if a train is parked at the next substation and is going in the same direction, then the passing rule applies instead (rule four).

Rule2: Whenever a fast train contends with a slow train for a resource, the fast train has higher priority than the slow train.

Rule3: Whenever two trains of the same type (fast or slow) contend for a resource, the train arriving first has higher priority than a train arriving later.

Rule4: A fast train can pass a slow train going the same direction at the next substation if a track to and a parking place at the next substation is available and two trains going in the opposite direction are not passing at any substation beyond the next substation.

Rule 5: If the number of the trains going in the same direction is larger than a given number, no more new train arrivals are allowed into the system from that direction (i.e., they wait at station A or E).

2.6 Effectiveness of Induced Rules

The regulator process monitors changes of knowledge-base facts to allocate system resources using rules. When the regulator is consulted, it executes the OpEM Expert System Controller Program to decide if each train waiting at a station can move.

After all train movement possibilities are decided, global rules are then used to select the train having

highest priority for each track. Figure 6 shows one of the global decision making rules.

```
CONSTRAINT_RULE069:IF
     BST^ Decision_E_A=Move_E_D AND
     E_A_Train_State=WT4 AND
     E_A_Train_Speed=Fast AND
     E_A_Train_Wait=Long
THEN
     Allocate_Track_4=Yes, CF=95.
```

Figure 6: A Global Decision Making Rule.

The BST^ command in rule069 causes the inference engine to search the knowledge-base for all instances of fact "Decision_E_A=Move_E_D." For each instance found, the remaining premise facts are tested. Of the instances, for all rules for this decision, with rule premise satisfied, the rule with highest confidence is selected. This rule implements decision policy rules two and three discussed above.

For the system under study, given assumptions are: (1) number of tracks per substation equals 2, (2) probability of a fast train equals 0.5, (3) average speed of fast train equals 100 miles per hour, and (4) average speed of slow train equals 50 miles per hour. For an optimal system, no waiting time occurs for a train at any substations. Under optimal conditions a slow train transits in two hours and a fast train in one hour.

Figure 7 shows train transit time as a function of train interarrival time. The smaller the interarrival time the more trains that are in the system contending

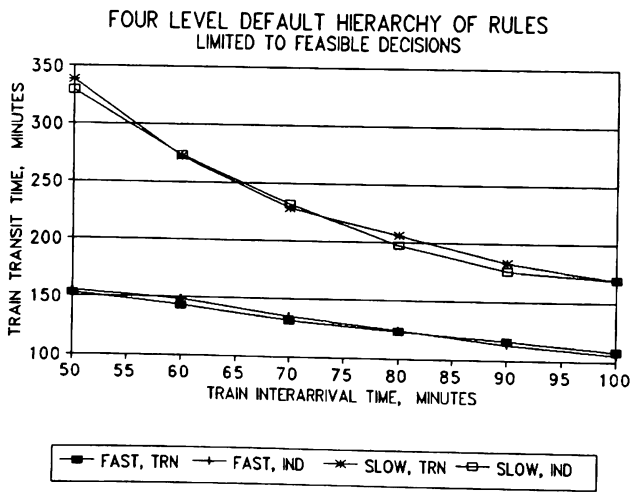

Figure 7. Effectiveness of training set and induced rules.

for resources. The figure shows that the policy of fast trains having higher priority has been implemented. The top line is for decisions made for slow trains, and the much lower line is for decisions made for fast trains. These results are superior to those reported in (Clymer 1992c) because more fast trains are in the system and learning is constrained to feasible decisions.

3 COMPARISON OF DECISION TOOLS

3.1 OpEM Expert System Controller

There are two kinds of objects in OpEM simulations: (a) entity or process objects that represent the state of the system and its environment and (b) symbolic objects that represent what is perceived, and thus known, about the system and its environment needed to make decisions.

Symbolic objects are usually described by an object-attribute-value triplet. The object part is the name of an entity or process. An example is a surface ship target, called Tgt. The attribute part is one of several object features. An example is ship to target range, called Rng. The OpEM expert system controller combines the object part and attribute part into the object name. For example, an object name could be TgtRng. The value part provides a value for an object name, forming an object-value couplet, or fact. For example, ship to target range is close, represented as "Tgt-Rng=Close", forms a fact.

Figure 8 shows a simple rule tree to assign a resource. The goal object to be pursued is Allocate_Res that can be given a value of either **yes** or **no**. Two rules are shown that connect symbolic objects Tgt_Needs_Res and Tgt_Priority to the goal. If the premise is true (i.e., value of Tgt_Needs_Res is **Yes** and value of Tgt_Priority is **High**), the conclusion is true for the rule on the left side of the figure and the goal object is given the value **yes**. If the premise is true (i.e., value of Tgt_Needs_Res is **No** or the value of

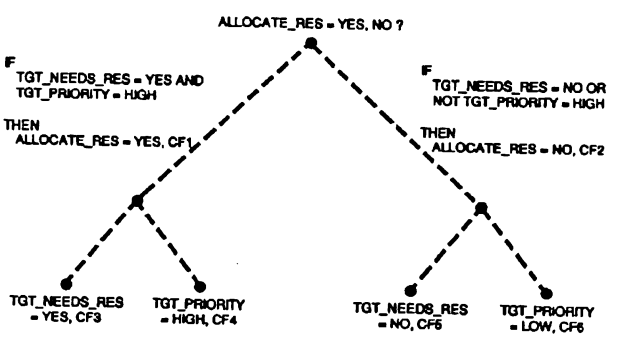

Figure 8: A Simple Rule Tree

Tgt_Priority is **Low**), the conclusion is true for the rule on the right side of the figure and the goal object is given the value **No**.

If the premise facts are fuzzy (Clymer 1992a, 1992b, Negoita 1985), both rules can produce conclusions. Confidence in a fuzzy fact such as "Tgt-Rng=Close" is a function of a variable X, the slant range between target and ship. Fuzzy fact functions used by controller all have the form 100/{1+[(1/A)(X-C)]**B}, shown in figure 9, where A, B, and C are all integer values with both A and C positive. The shape of the fuzzy membership function can be defined by specifying A, B, and C. "A" is the spread of the membership function, "C +A" is the mean value of the function (i.e., where the evaluated certainty factor equals 50), and "B" controls the slope. C is kept zero unless a symmetrical function is desired.

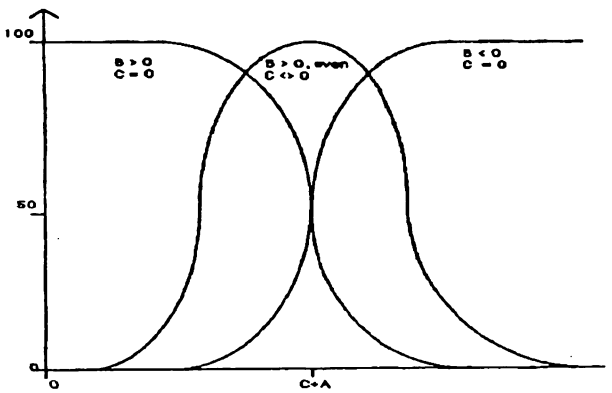

Figure 9: Fuzzy Fact Set Functions

If B is a positive integer, a typical S-function results. An inverted S-function results when B is negative. If B is even, the function becomes a rounded peak about the mean value of (C +A). In this case, C should not be zero and should be large enough to allow the membership function to be symmetric everywhere.

A more complex rule tree is shown in Figure 10. This tree has intermediate rules that give values to decision objects. The top of a rule tree is the goal object. All rules having the goal object name in their conclusion form the top branches of the tree. At each level rules connect premises to conclusions. Premises of rules contain either knowledge-base facts or decision facts that are inferred facts. In addition, if each case is covered by a set of rules of increasing specificity, a default rule hierarchy is formed (Clymer 1990, Holland et al 1985) that is useful in adaptive learning.

The OpEM Pascal Simulation Tool Kit (Clymer 1990a, 1990b) routines are executed by a simulation

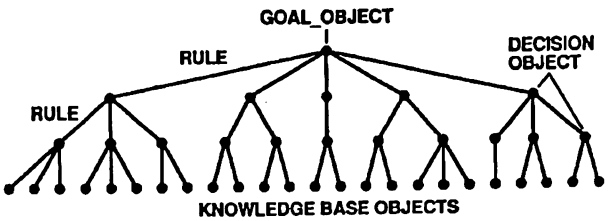

Figure 10: A More Complex Rule Tree

program to generate event-state sequences (timelines) for various situations that can occur in a scenario. The OpEM Expert System Controller (Clymer 1989, 1990c, 1992a, 1992b, 1992c) has interfaces with these simulation routines to resolve decision conflicts and choose the best (most effective) decision. Given the decision fact returned by controller, an event or chain of events, each event represented by an event subroutine, is executed to change appropriate state dimensions to execute the transition. The rules used by controller to make these decisions can be discovered using the OpEM Pascal Induction Program.

The OpEM Pascal Induction Program (Clymer 1992b, 1992c) receives as input a case file generated by an OpEM discrete event simulation program. Each case consists of a decision fact (i.e., the decision made) plus all knowledge-base facts available for this decision (i.e., the decision context). The OpEM induction program analyzes this set of cases and produces a set of rules that decides all of these cases correctly. The OpEM induction program uses a top-down, combinatorial search for concepts based on an ordered list of decision facts for each decision, and it is similar to the "Greedy Algorithm" discussed in (Haussler 1988).

Hypotheses are candidate rules of the form "IF Fact THEN Decision". Hypotheses may be more complicated and several facts may be present in the premise, thus more complicated hypotheses may be tested of the conjunctive form "IF (Fact_A AND Fact_B AND.....) THEN Decision_A". The set of all possible rules that can be formed from the available facts (called a decision list) is called the hypothesis space (Haussler 1988).

Cases are searched for associations between premise facts and decision facts. The first step is to count the number of times each premise fact is associated with each decision fact. For each decision fact, a hypothesis is then formed relating it to the single most frequently occurring premise fact. If a hypothesis is consistent (Haussler 1988, Michalski 1983), covers positive cases and no negative cases, for the maximum number of cases not yet covered it is accepted as a rule.

If not, another hypothesis is formed combining the most frequently occurring premise fact with the next most frequently occurring premise fact and this new hypothesis is tested to determine if it is an acceptable rule. If it is not an acceptable rule, the process continues, each time the next most frequently occurring premise fact is added to the premise until an acceptable rule is found. The search proceeds from the most general hypotheses (one fact premise) to more specific hypotheses until a consistent rule is found. The algorithm considers a set of competing hypotheses, generated as just described, and selects rules that cover the most cases. Cases are removed from consideration after they have been covered by a specified number of rules, forming a default hierarchy.

A rule bid is a function of premise fact confidences and rule strength. The rule with the highest bid calculation usually makes the decision. The OpEM expert system controller has an adaptive mode (Clymer 1992a, 1992b), based on the "bucket brigade" algorithm (Holland et al 1985), that automatically determines rule strength, measuring how well a rule contributes to system success. If a set of rules of increasing specificity is generated to cover each case, a default hierarchy of rules results as discussed above. If the adaptive mode reduces rule strength greatly because the rule is too general (too many new cases occur where the rule fails), it is important to have more specific rules in version space (Haussler 1988) as backup. Having a default hierarchy of rules allows the best rules to be selected through adaptive learning. This is important because only a subset of cases is ever available in the training set for induction.

The OpEM Adaptive Expert System Controller, when in inductive and adaptive mode, writes cases to a case file each time a rule fails to decide correctly or no rules are able to decide. Induction based on the new cases, combined with the previous set, produces a more robust rule set. This mode of operation, similar to incremental induction of concepts (Genneri, Langley, and Fisher 1990) or classifier systems (Goldberg 1989, Grefenstette 1988) but done off-line, continues until high confidence rule failures no longer occur. This mode is called off-line incremental induction here.

As an example of the improvement that can occur, the initial case file for a sonar classification problem (Clymer 1992b) was based on decisions for only ten ships. Rules induced using this case file were applied to classify 250 ships. Cases corresponding to rule failures were added to the initial case file and rules again induced in a second iteration. The second set of rules classified 500 ships running in adaptive mode. The adapted rules classified a thousand ships without error in the no noise case. When noise was present, the adapted rules achieved optimal decision making performance.

3.2 Classifier Systems

In classifier systems (Goldberg 1989, Grefenstette 1988, Holland et al 1985), the premise or conclusion of a rule can be expressed as a vector of numbers called a message. Each number in the premise message indicates if a particular fact is included, not included, or included/not-included (don't care) in the premise. Each number in the conclusion message indicates if a fact is included, not included, or passed through from the premise message. A rule posts a conclusion message in the knowledge base if its premise message is found in the knowledge base (i.e., rule fires) and if it wins the bidding contest with other fired rules.

An example is a small animal that seeks out and consumes insects but runs and hides when a hawk appears. A classifier system generates a sequence of effecter messages that causes the animal to scan its environment. Messages from the environment, received while looking around, indicate positions of insects. The classifier system next generates a sequence of effecter messages that causes the animal to close on an insect and grab it.

System operation is modeled using uniform time steps between messages resulting in changes of system state that can be a physical change in the system or a change in knowledge about the environment. Messages are received from the environment, a sequence of rules post messages in pursuit of system goals, and messages are sent to system effecters to implement system goals. In the small animal example, the goals are to eat insects and avoid hawks.

Learning occurs by adapting rule strengths to modify the bidding contest outcomes and by generating new rules. New rules are generated using genetic operators to decide premise and conclusion fact changes. Through rule strength modification, bad rules generated are weakened and eventually eliminated by being punished when system goals are not achieved. Good rules are strengthened and eventually accepted by being rewarded when system goals are achieved. Rule credit assignment is difficult in context sensitive situations, however (Grefenstette 1988). The search through hypothesis space, resulting in convergence to an optimal set of rules to make system decisions, is more rapidly accomplished by considering a large number of alternate hypotheses in parallel, called a beam search. Rule bids are used to compute the probability of selecting a rule, giving all alternative rules a chance of being selected. A beam search reduces the likelihood that a

search for rules halts at a local optima, called the "hill climbing problem."

3.3 Neural Networks

Figure 11 shows a two-level backpropagation neural network to predict position of a target **Y** given a sequence of observations of target **X**. Each observation consists of a position of the sensor along its path and the associated angle from sensor to target.

A set of inputs **X** is applied to the first layer of the

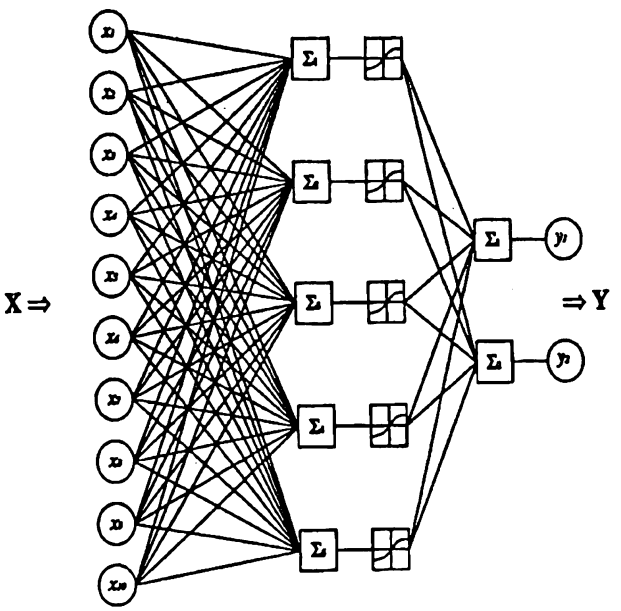

Figure 11: A Neural Network

neural network. Each of these inputs is multiplied by a weight, and the products summed. This summation of products is termed Σ_i and must be calculated for each neuron i, where i equals 1 to 5, in the figure. After Σ_i is calculated, an activation function **F** is applied to modify Σ_i, thereby producing an output signal. The output signal of the first layer is the input to the next (output) layer as shown in figure 11.

The objective of training the network is to adjust the weights so application of a set of inputs produces a desired set of outputs. See Wasserman (1989) for details on how training is accomplished. For the passive ranging problem, a large number of target-sensor encounters were simulated to provide a training set of input-output pairs to train the network. If sensor-target geometry is restricted to a 20-80 degree look angle, the neural network learned to predict target position.

3.4 Case-based Reasoning

Figure 12 shows a case-based planning function (Hammond 1989, Slade 1988) that accepts input on the goals to be satisfied and features of the situation covered and then generates a plan of action. A plan is a sequence of actions that satisfies a set of goals for a system given a particular starting situation. The case-based planner shown is a learning system that learns by remembering plans that avoid problems, features that predict problems, and repairs that have to be made if those problems arise again in another situation. It makes use of feedback about plan mission effectiveness to learn to avoid planning problems and to repair faulty plans that occur.

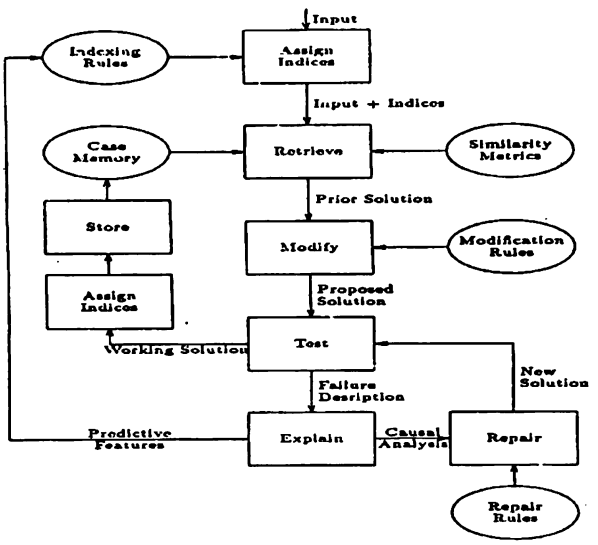

Figure 12: A Case-based Reasoning System

The assign indices function accepts input on the goals of a plan and features of the situation. It applies rules to such information to generate indices that characterize the particular situation such as problems, negative interactions between plan steps, that could arise in the new situations.

Plans stored in memory are indexed by goals they satisfy, features of the situation, and by the problems they avoid and other characteristics. Therefore, the retriever function receives these three types of information as input and then uses them to locate the best plan. A similarity metric is used to judge the similarity of goals when determining partial matches. A plan may partially satisfy a goal (build a chair) by satisfying a more general statement of the goal (build furniture). A value hierarchy of goals, where the highest value goals

are the most difficult to modify when not satisfied, is used to judge the relative utility of plans with respect to a set of goals.

The plan modifier function receives as input an old plan suggested by the retriever to cover the new situation and modifies it to satisfy any goals not satisfied completely. To alter old plans to meet new goals and situations, the modifier needs a set of modification rules, critics with knowledge of goal specific requirements, and general plan specifications. Modification rules specify what steps to add to particular plans, given a particular goal to satisfy. Information in the form of special purpose critics, tailor the general modifications of a plan to the specific needs of the items required to achieve particular goals.

The new plan is executed, and it is evaluated to determine how well it performed. The test function can be done in the real world or using a simulation program. If a proposed plan achieves all of its goals, it is assigned indices and stored in memory. When a plan fails to achieve all its goals, it must be repaired so that the planning system can learn from its mistakes.

To place new plans in memory, the assign indices and store functions index new plans under the same goals, features, and problems that the retriever uses to find plans. The goals that are used to index plans, however, are generalized so that the plans can be found in situations that are similar if not identical to those in which they were originally constructed. Generalization allows the planner to reason using analogy.

The inputs to the explain and repair functions are the faulty plan and some description of the fault. A fault is either a desired state that was not achieved or an undesired state that arose during plan execution. To repair failed plans and describe them to the assign indices and store functions, the repairer function requires a vocabulary of plan failures and repair strategies that are indexed by the vocabulary. The repaired plan is sent back to be tested

To anticipate failures in the future, the cause of a plan failure must be determined. To decide which features in a situation are to blame for a failure, the explain function needs to be able to build causal explanations of planning failures and to mark the states and steps that lead to the failures as predictive of them. The more extensive its vocabulary for this description, the more exact its credit assignment will be.

Domain knowledge required for the goal value hierarchy, similarity metrics, modifier rules, critics and plan specifications, failure vocabulary, and repair strategies is extracted from an expert. The expert evaluates a plan as good or bad and, when a plan is evaluated bad, provides a causal explanation of why the plan failed. A case-based planner is best applied in weak theory domains when the number of training cases is small and maximum information (classification categories such as good and bad and domain knowledge used to build plans) must be extracted from an expert.

3.5 Comparison

Neural networks are conceptually similar to a rule based tool, such as a classifier system or the OpEM expert system controller. Their network connections accomplish the same function as rules.

Neural networks have the distinct advantage of highly parallel execution for real time applications; however, on a single CPU, backpropagation neural networks required 500 times as long to learn and 10 times as long to classify as symbolic methods when applied to four learning problems (Mooney et al 1989). Counterpropagation networks (Wasserman 1989) are reported to be much faster, however. Once trained, neural networks decide with about the same accuracy as symbolic methods (Mooney et al 1989, Weiss and Kapouleas 1989). This is further confirmed by experiments where air traffic control decisions made using the OpEM expert system controller (Clymer 1992a, 1992b) were simulated and a backpropagation neural network was trained. The neural network controller had comparable performance to the expert system controller.

The main difference between neural networks and rule based tools, such as a classifier system or the OpEM expert system controller, is that these rule based tools can add new rules to a rule tree. A neural network cannot add new connections or layers to itself. It can only adapt connections in layers initially defined. Further, according to (Mooney et al 1989, Weiss and Kapouleas 1989), a significant amount of time may be required to determine the optimal number of neurons in a hidden layer. The OpEM expert system controller and classifier systems both incrementally add rules to their rule bases.

Learning speed and control system performance seem to be a function of the amount of domain knowledge used to guide the search through version space to find the best rules to cover each control situation. Classifier systems use genetic algorithms to implement a generate and test search through version space using little domain knowledge (Holland et al 1985). Case-based learners acquire very specific domain knowledge about each situation, from an expert, to guide its search for rules. According to Ellman (1989), applying acquired domain knowledge does greatly increase learning speed relative to other induc-

tive methods that do not; however, too much reliance on domain knowledge seems to limit what the system can learn. Classifier systems seem capable of going beyond available domain knowledge to discover new concepts useful in system control. Thus, there seems to be a tradeoff between supervised learning, that uses specific domain knowledge to improve learning speed, and unsupervised learning that can discover more effective rules that transcend such available domain knowledge.

3.6 Conclusion

Two basic problems are encountered when discovering rules to improve system control decisions. These are credit assignment and rule generation.

Credit assignment modifies the strengths of rules that are used to make decisions in a sequence of events leading to either system mission success or failure. If the mission fails, all rules contributing to the failure are punished. If the mission succeeds, all rules contributing to the success are rewarded.

One purpose of credit assignment is to optimize decision making given a set of rules by altering the bidding competition among the rules. The sonar classification scenario discussed above is one example. Another purpose of credit assignment is to identify contexts where decision making could be improved by generation of better rules.

The credit assignment problem occurs because of context sensitive interactions among related decisions in a timeline. This problem can be solved by evaluating related decisions using domain knowledge. For example, credit assignment is modified to consider whether a resource allocation decision followed priority policy, stated in the form of contraint rules.

Contraint rules are also used to define when a decision **can** be made. Thus, domain knowledge is used to restrict the search for new rules to discovering when a decision **should** be made.

Credit assignment guides the search for new rules as follows: (1) rules that are punished in some situations and rewared in others are too general and must be made more specific and (2) rules that have high strengths but do not fire often are too specific.

Thus, adaptive rule strength modification, found in classifier systems, should be used to maintain alternative sets of rules that compete (a beam search), and timelines should be analyzed to identify good and bad cases of rule application using domain knowledge about context sensitive systems to guide the search for better rules. Such an approach would be a balance between classifier, case-based, and explanation-based learning systems.

4 SUMMARY

Decision making in context sensitive systems seems best accomplished using a combination of the credit assignment and parallel (beam) search from classifier systems; the rule-based reactive decision making approach from the OpEM Controller; and the critic process and associative plan memory from case-based and explanation-based reasoning approaches. Each of these approaches have weaknesses by themselves, but combining the strengths of each individual approach, where the other approaches are weak, a much more robust learning system can be expected to result.

REFERENCES

Angluin, D. and C. Smith, (1983). Inductive Inference: Theory and Methods, *Computing Surveys*, Association for Computing Machinery, Volume 15, Number 3, pages 237-269.

Buchanan, B.G., Sullivan, J., Cheng, T-P, and S.H. Clearwater, (1988). "Simulation-Assisted Inductive Learning," In *Proceedings- 7th National Conference on Artificial Intelligence*, Minneapolis,MN, August 22-26,1988.

Cl ymer, J.R., (1989). "OpEM Expert System Controller," In *Simulation and AI, 1989,* San Diego, CA: The Society of Computer Simulation International, Volume 20, Number 3, pages 20-26.

Cl ymer, J.R., (1990a). *Systems Analysis Using Simulation and Markov Models*, Englewood Cliffs, N.J.: Prentice-Hall, Inc.

Cl ymer, J.R., Corey, P.D., and N. Nili, (1990b). "Operational Evaluation Modeling," In *Simulation*, San Diego, CA: The Society of Computer Simulation International, December 1990 issue, pages 261-270.

Cl ymer, J.R., (1990c). "System Design Using OpEM Inductive/Adaptive Expert System Controller," In *IASTED International Journal of Modeling & Simulation*, Volume 10, Number 4, pages 129-136.

Cl ymer, J.R. and D. Hernandez, (1991). "OpEM Distributed Simulation," In *Simulation*, San Diego, CA: The Society of Computer Simulation International, December 1991 issue, pages 395-404.

Cl ymer, J.R., Corey, P.D., and J. Gardner, (1992a). "Discrete Event Fuzzy Airport Control," In *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, Number 2, March-April 1992, pages 343-351.

Clymer, J.R. and P.D. Corey, (1992b). *Simulation of Intelligent Decision Making in Context Sensitive Systems*, Placentia, CA: John R. Clymer and Associates, Inc.

Clymer, J.R., D.J. Cheng, and D. Hernandez, (1992c). "Induction of Decision Making Rules for Context Sensitive Systems," In *Simulation*, San Diego, CA: The Society of Computer Simulation International, September 1992 issue, pages 198-206.

Corey, P.D., and J.R. Clymer, (1991). "Discrete Event Simulation of Object Movement and Interactions," In *Simulation*, San Diego, CA: The Society of Computer Simulation International, March 1991 issue, pages 167-174.

Ellman, T. (1989). "Explanation-Based Learning: A Survey of Programs and Perspectives," ACM Computing Surveys, Volume 21, Number 2, June issue, pages 163-221.

Gennari, J.H., Langley, P., and D. Fisher, (1990). "Models of Incremental Concept Formation," In *Machine Learning: Paradigms and Methods*, J. Carbonell ed., Cambridge, Mass: MIT Press, pages 11-61.

Goldberg, D.E., (1989). *Genetic Algorithms: in Search, Optimization, & Machine Learning*, New York, NY: Addison-Wesley.

Grefenstette, J.J., (1988). "Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms," In *Machine Learning*, 3, Boston, MA: Khwuwer Academic Publishers, pages 225-245.

Gray, N.A.B, (1990). "Capturing Knowledge Through Top-Down Induction of Decision trees," *IEEE Expert*, July 1990 issue, pages 41-50.

Hammond, K.J., (1989). "Chef," In C.K. Riesbeck and R.C. Schank Eds. *Inside Case-Based Reasoning*, Hilldale, NJ:Erlbaum.

Haussler, D., (1988). "Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework," In *Artificial Intelligence*, 36, pp 177-222.

Holland, J.H., Holyoak, K.J., Nisbett, R.E., and P.R. Thagard, (1986). *Induction: Process of Induction, Learning, and Discovery*. Cambridge, Mass: The MIT Press.

Michalski, R.S., (1983). "A Theory and Methodology of Inductive Learning." In *Artificial Intelligence*, 20, pages 111-116.

Mitchell, T.M., (1980). "The Need for Biases in Learning Generalizations," Technical Report Number CBM-TR-117, New Brunswick, NJ: Rutgers University, Department of Computer Science.

Mooney, R., Shavlik, J., Towell, G., and A. Gove, (1989). "An Experimental Comparision of Symbolic and Connectionist Learning Algorithms," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann, pages 775-780.

Negoita, C., (1985). *Expert Systems and Fuzzy Systems*. The Benjamin /Cummings Publishing Company.

Quinlan,J.R., (1983). "Learning Efficient Classification Procedures and their Application to Chess End Games." *Machine Learning: An Artificial Intelligence Approach*, Los Altos,CA: Morgan Kaufmann Publishers, pages 463-482.

Quinlan,J.R., (1986). "Induction of Decision Trees," In *Machine Learning*, 1, Boston, MA: Khwuwer Academic Publishers, pages 81-106.

Reddy, Y., et al (1986). "The Knowledge-Based Simulation System," In *IEEE Software*, Volume 3, Number 2, pages 25-37.

Slade, S., (1988). "Case-Based Reasoning: A Research Paradigm," Yale University, Department of Computer Science report, AD-A202 363.

Wasserman, P.D., (1989). *Neural Computing: Theory and Practice*, New York, NY: Van Nostrand Reinhold.

Weiss,S.M., and I. Kapouleas, (1989). "An Empirical Comparision of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann, pages 781-787.

AUTHOR BIOGRAPHY

JOHN R. CLYMER is a professor of electrical engineering at California State University Fullerton (CSUF) and consults for Rockwell International, Naval Surface Warfare Center, and others in the area of systems engineering. In addition to consulting, he presents intensive short cources on system engineering at various locations around the United States. His teaching assignments have included computer logic design, assembly language programming, numerical analysis, linear systems analysis, continuous systems simulation, operational analysis and optimization, mathematical programming, and artificial intelligence. Dr. Clymer's reseach interests include distributed simulation, combined discrete event and continuous simulation, computer aided systems engineering tools, and intelligent decision making systems, and he is a founding member of the Applied Research Center for Systems Science at CSUF. Dr. Clymer is currently the Chair of the IEEE Orange County CA Section.