Tower of Hanoi with Connectionist Networks: Learning New Features

Charles W. Anderson (CWA%GTE.COM@RELAY.CS.NET)

Self-Improving Systems Department, GTE Laboratories Incorporated, Waltham, MA 02254 U.S.A. (617) 466-4157

Abstract

A connectionist system previously used to solve the numerical control task of balancing a pole (Anderson, 1987) is applied with little modification to a three-disk, Tower of Hanoi puzzle whose solution is typically represented symbolically (as production rules, for example). The connectionist system consists of two networks: an evaluation network that learns an evaluation function of states, and an action network that learns heuristics for searching for good actions. The initial state representation is insufficient—new features must be learned to form a useful evaluation function. Comparisons of methodology are made with Langley's (1985) adaptive production system, SAGE.

1 The Tower of Hanoi Puzzle

The Tower of Hanoi puzzle is popular for research in problem-solving. The small number of states facilitates the analysis of solution strategies and simplifies its simulation, yet the puzzle is challenging. Human strategies for solving the Tower of Hanoi puzzle have been analyzed (Luger, 1976) and modeled (Anzai and Simon, 1979). Amarel (1981) used the Tower of Hanoi puzzle as a vehicle for studying shifts of representations to forms of increasing efficiency for the discovery of a problem's solution.

The state of the Tower of Hanoi puzzle can be represented in a number of ways. A common representation is one used by Nilsson (1971), for which the pegs are numbered 1, 2, and 3, and the disks are labeled A, B, and C, where Disk A is the smallest disk and C is the largest. A particular state is represented by the peg numbers where each disk resides, listed for disk C, then disk B and disk A. As pictured in Figure 1, the initial state of the puzzle is (111), and the objective is to achieve state (333) by applying a sequence of actions. The only legal actions are movements of the top-most disk from one peg to another, with the restriction that a disk may never be placed upon a smaller disk. An action may be represented as a source peg and destination peg, so the transformation of state (111) to state (112) is performed by the action of moving the top-most (smallest) disk from peg 1 to peg 2, represented by Action 1-2. For the three-peg puzzle, only six actions exist: 1-2, 1-3, 2-1, 2-3, 3-1, and 3-2.

The states of the puzzle plus the transitions between the states corresponding to the legal actions form the puzzle's state transition graph, as shown in Figure 2. To evaluate a human's or machine's ability to improve its search strategies on the Tower of Hanoi puzzle, we measure the number of actions in the solution path—a route through the state transition graph from the initial to the goal state—the minimum length path being the objective. For the three-disk puzzle, the minimum-length solution path has seven actions and is the straight path down the right side of the state transition graph. Finding the shortest solution path is confounded by the large number of possible solution paths and by the presence of many loops, or cycles, in the state transition graph.

The Tower of Hanoi puzzle is a good test of the multilayer connectionist system developed in Anderson (1987), particularly of its ability to generate new features, for the following reason. A useful evaluation function must map states to values that indicate the states' closeness to the goal node. For the state representation described above, this mapping can be visualized by considering the peg numbers to be dimensions of a three-dimensional state space, as shown in

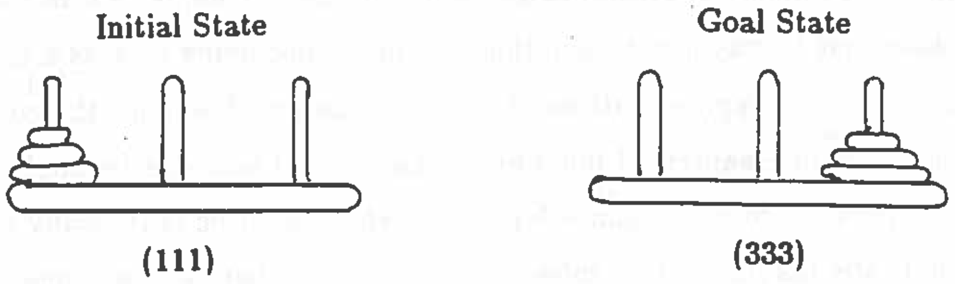

Figure 1: Initial and Goal States of the Tower of Hanoi Puzzle

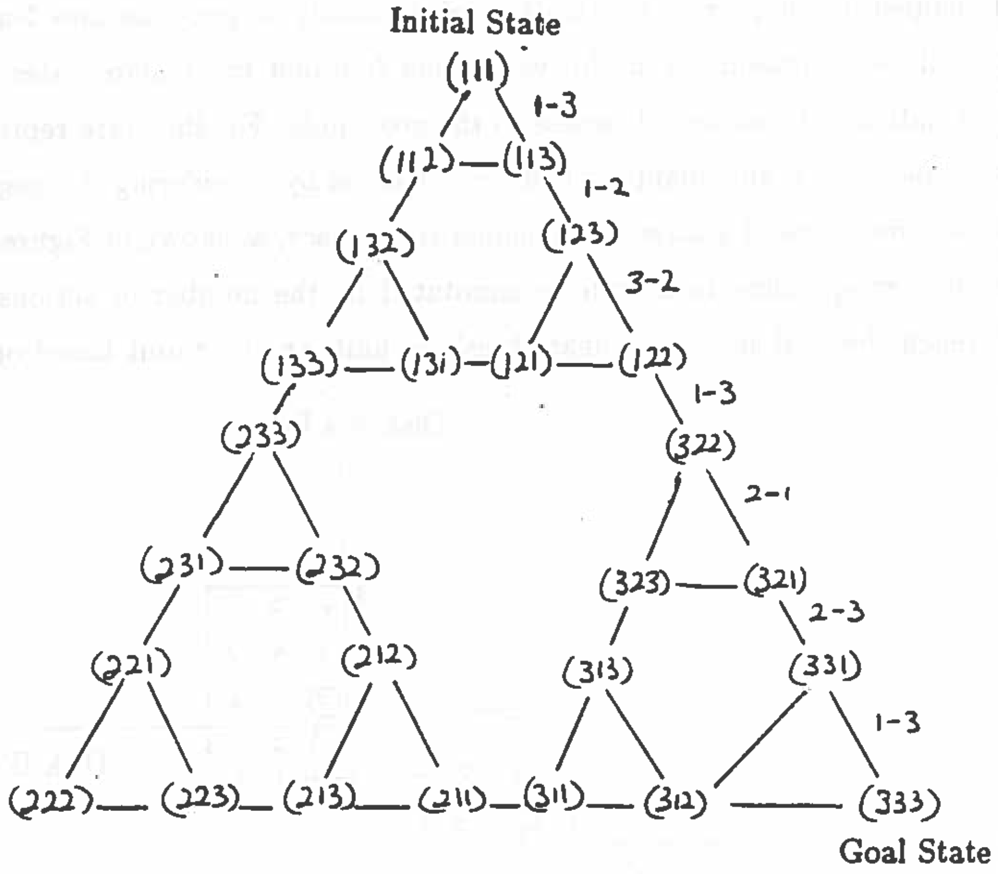

Figure 2: State Transition Graph for Three-Disk Tower of Hanoi Puzzle (Adapted from Nilsson, 1971)

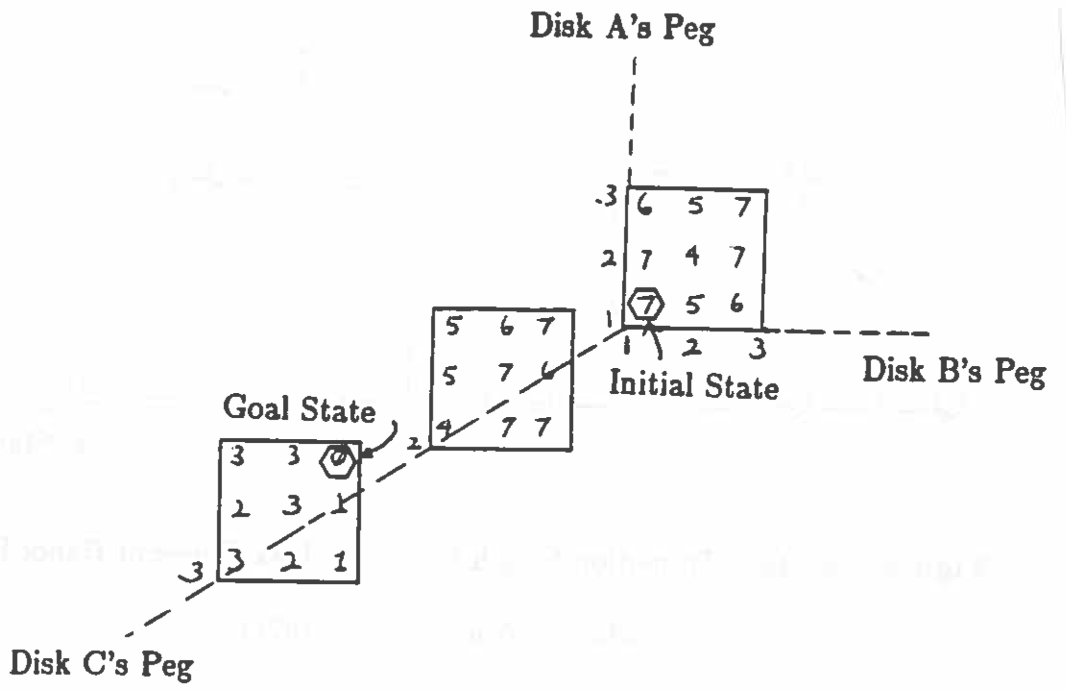

Figure 3: 3-Dimensional State Space and Desired Evaluations

Figure 3. Each point corresponding to a state is annotated by the number of actions required to reach the goal state. A linear threshold unit, or other unit based on a linear weighted sum of its inputs, cannot make the necessary discriminations between states to form a good evaluation function. New features must be formed by a layer of hidden units that carve up the state space into a representation for which the linear output unit can learn a good evaluation function. For the experiments described here, the representation was simplified somewhat, as described later, to reduce the time required to learn the solution. The fact that new features are still required is shown by the inability of a one-layer system to learn the minimal solution path.

The formation of useful search heuristics for the Tower of Hanoi is less complicated. A small set of rather simple heuristics can constrain the search to exactly the correct actions (Anzai and Simon, 1979; Langley, 1985). For example, many alternatives are removed by a rule stating that it is undesirable to apply the inverse of an action. The action network used to learn search heuristics in the following experiments is single-layered, and did successfully learn the minimal solution path. The representation provided to the network is sufficient—new features are not required. The simplicity of the search heuristics does not lessen the need for a good evaluation function; credit must be assigned correctly in order for the heuristics to be learned.

2 Experiments

As in the pole-balancing experiments (Anderson, 1987), the performance of a one-layer connectionist system is compared to the performance of a two-layer system to demonstrate the development of useful features. The learning algorithms and the networks are very similar to those used for the pole-balancing task, with small modifications to the network structure and the algorithms. Before describing these modifications, the state and action representations are discussed. Then the results of experiments with the one and two-layer networks are presented.

2.1 Representation of States and Actions

As mentioned above, the state representation consisting of the three peg-numbers corresponding to each disk results in a very complex mapping from states to evaluations. Although in principle the connectionist systems used here should be able to learn this mapping, we wished to simplify the task somewhat to reduce the simulation time required for the experiments.

The state representation used in the following experiments is composed of nine binary digits. The first three digits encode Disk C's peg number, the second encode Disk B's peg, and the third set of three digits encode Disk A's peg. Peg 1 is encoded as 100, Peg 2 as 010, and Peg 3 as 001. For example, state (111) is represented as (100 100 100), and state (123) is represented as (100 010 001).

The output of the action network represents an action by a six-component, standard unit basis vector, where the components correspond to actions 1-2, 1-3, 2-1, 2-3, 3-1, and 3-2, respectively; Action 1-2 is encoded as (100000), Action 1-3 is (010000), and so on.

Both the evaluation network and the action network receive the representation of the state. This completely defines the input to the evaluation network, but additional terms are presented to the action network. We wished to investigate the ability of the connectionist network to learn search heuristics similar to the rules developed by Langley's SAGE system. As mentioned above, one such rule is to never apply the inverse of the previous action. In order to learn such an association between the previous action and the current action, the previous action must be provided as input to the action network. Another rule learned by SAGE is to not apply an action that returns the puzzle to a state that was visited two steps ago. This avoids the three-step loops around the smallest triangles in the state transition graph (Figure 2). Rather than providing past states as input, we chose to present the action taken two-steps ago, in addition to the previous action. The previous two actions along with the current state provide enough information to identify the state visited two steps ago, although our results suggest that hidden units are needed to overcome the linearity of the output unit, perhaps by forming a conjunction of the previous two actions. This possibility was not investigated.

2.2 Credit Assignment

The evaluation network learns an evaluation function, but in its initial state the network implements a meaningless function that evaluates all states approximately equally since its weights are initialized to small random values. With experience it converges on a function that predicts the occurrence of reinforcements.

The most significant reinforcement occurs whenever the goal state is entered. A reinforcement value, labeled $r[t]$, of 1 is presented for the time step at which the goal state (333) is entered. Recall that for the pole-balancing task, the goal is to avoid certain states for as long as possible, and $r[t]$ was set to -1 upon entering those states. The Tower of Hanoi task could be solved (by a two-layer network) with only this final reinforcement, but two additional reinforcements are provided for the following reasons.

If the action probabilities converge too quickly, due to a large value for the parameter ρ, a solution path of longer than minimum length will probably be learned. For example, say the learned solution path is of length eight, one step longer than the minimum number, due to the incorrect Action 1-2 being taken from the starting state (111). If this action is always chosen over the correct Action 1-3, then an evaluation function tailored to this particular solution path will be learned. To avoid this, a second reinforcement signal is presented having a constant value of -0.1 for all non-goal states. In this way, a shorter solution path results in a higher total reinforcement than does a longer solution path. This parallels the role of Langley's heuristic for judging shorter paths between two states as more desirable.

The third reinforcement is a value of -1 presented whenever a two-step loop occurs, i.e., when the current action reverses the effect of the previous action. The random search initially followed by the action network results in many two-step loops (and longer loops), thus many steps elapse before the goal is discovered. The large negative reinforcement results in a significant decrease in the probability of selecting the action that is the inverse of the previous action. As shown later, this must be learned for each action—the concept of inverse actions is not known to the system, so generalizing across actions is not possible. This reinforcement is not presented to the evaluation network, enabling a large negative reinforcement to be used without decreasing the evaluation for the corresponding state. The large negative reinforcement is not meant to indicate that a state is bad, only that the action was bad. The selection of the inverse action should be discouraged, but not necessarily the visitation of the state.

This third type of reinforcement is not necessary for the system to learn the puzzle's solution. It was included for two reasons: it significantly reduces the number of search steps during the early stages of learning, and it demonstrates how domain knowledge, such as the undesirability of one-step loops, can be added by altering the reinforcement function.

2.3 Interaction between Learning System and Puzzle

The input to the evaluation network is composed of the nine binary digits that encode the state of the puzzle, plus a constant component with a value of 0.5. Specifically, the input terms, $x_i[t]$, for the state at time t are given by:

$$\begin{aligned} x_0[t] &= 0.5, \\ x_1[t] &= \begin{cases} 1, & \text{if Disk C is on Peg 1 at time } t; \\ 0, & \text{otherwise,} \end{cases} \\ x_2[t] &= \begin{cases} 1, & \text{if Disk C is on Peg 2 at time } t; \\ 0, & \text{otherwise,} \end{cases} \\ x_3[t] &= \begin{cases} 1, & \text{if Disk C is on Peg 3 at time } t; \\ 0, & \text{otherwise,} \end{cases} \end{aligned}$$

and similarly for $x_4[t]$, $x_5[t]$, $x_6[t]$ and Disk B, and for $x_7[t]$, $x_8[t]$, $x_9[t]$ and Disk A. After the goal state is reached and at the start of every run, the state is set to (111), so the input becomes (100 100 100) disregarding the constant input.

The input to the action network consists of these terms and, in addition, the previous two actions. The action at time t is encoded by six binary components, $a_1[t], a_2[t], \ldots, a_6[t]$, so the additional input terms are defined as:

$$\begin{aligned} x_{10}[t] &= a_6[t-2], \\ &\vdots \\ x_{15}[t] &= a_1[t-2], \end{aligned}$$

and

$$\begin{aligned} x_{16}[t] &= a_6[t-1], \\ &\vdots \\ x_{21}[t] &= a_1[t-1]. \end{aligned}$$

The possible values of reinforcement are defined as follows. The value of $r[t]$ includes just the first two kinds of reinforcement, while the one-step loop penalty is given by $r_{\text{loop}}[t]$ to distinguish

t	State	Action	State $x_1 \ldots x_9$	Delayed Actions $x_{10} \ldots x_{15}$	$x_{16} \ldots x_{21}$	Action $a_1 \ldots a_6$	Evaluation $p[t]$	Reinforcement $r[t]$
1	111	1-3	100100100	000000	000000	010000	−0.32	−0.1
2	113	1-2	100100001	000000	010000	100000	−0.17	−0.1
3	123	3-2	100010001	010000	100000	000001	−0.14	−0.1
4	122	1-3	100010010	100000	000001	010000	0.10	−0.1
5	322	2-1	001010010	000001	010000	001000	0.35	−0.1
6	321	2-3	001010100	010000	001000	000100	0.51	−0.1
7	331	1-3	001001100	001000	000100	010000	0.75	1.0
8	333							

Table 1: Interaction of Learning System and Tower of Hanoi Puzzle

between the reinforcements that are and are not presented to the evaluation network:

$$r[t] = \begin{cases} 1.0, & \text{if state at time } t \text{ is } (333); \\ -0.1, & \text{otherwise.} \end{cases}$$

$$r_{\text{loop}}[t] = \begin{cases} -1.0, & \text{if state at } t \text{ equals state at } t-2; \\ 0.0, & \text{otherwise.} \end{cases}$$

An example of the interaction between the learning system and the Tower of Hanoi puzzle is shown in Table 1. The sequence is taken from a successful experiment, after the networks have learned good evaluation and action functions. The correct action is taken on each step, and the evaluation, p, increases as fewer steps remain to reach the goal state.

2.4 Results of One-Layer Experiments

The only modification to the learning algorithms used for the pole-balancing task involves the equation for updating the weights of the action network. The reinforcement signal for one-step loops, r_{loop}, is added as follows:

$$w_{i,j}[t] = w_{i,j}[t-1] + \rho\left(r_{\text{loop}}[t] + \hat{r}[t]\right)\Big(a_j[t-1] - E\{a_j[t-1] | w; x\}\Big)\, x_i[t-1].$$

As in the pole-balancing experiments, two performance measures were used to select the best values for the parameters of the weight-update equations. A measure of cumulative performance throughout a run is provided by the number of trials (achievements of the goal state) averaged over all runs for a given set of parameter values. The second performance measure is the average over all runs of the number of steps in the last trial~~, or the preceding trial, whichever is smaller~~.

The final performance level averaged over 5 runs of 50,000 steps each was used to select the best of approximately 20 sets of parameter values, differing in ρ and β, leaving $\gamma = 0.9$. The best values were $\beta = 0.100$, $\rho = 0.01$, $\gamma = 0.9$. These values were used in a longer experiment of 10 runs of 100,000 steps each, resulting in the number of trials and last trial lengths shown in Figure 2. The average number of trials is $3,145$. From the lengths of the last trial for each run we see that the minimal solution path was not learned in any run—all trials are longer than seven steps. Run 7 resulted in a last trial of length nine, but it wasn't determined whether the path taken on the final trial would be reliably followed for subsequent trials. The low number of total trials for Run 7 indicates that paths longer than nine steps are likely.

Run	Trials	Last Trial
1	2,911	28
2	2,877	23
3	2,884	19
4	2,893	65
5	2,686	651
6	2,928	25
7	4,481	9
8	2,902	25
9	3,951	38
10	2,940	41

Table 2: Results of One-Layer System

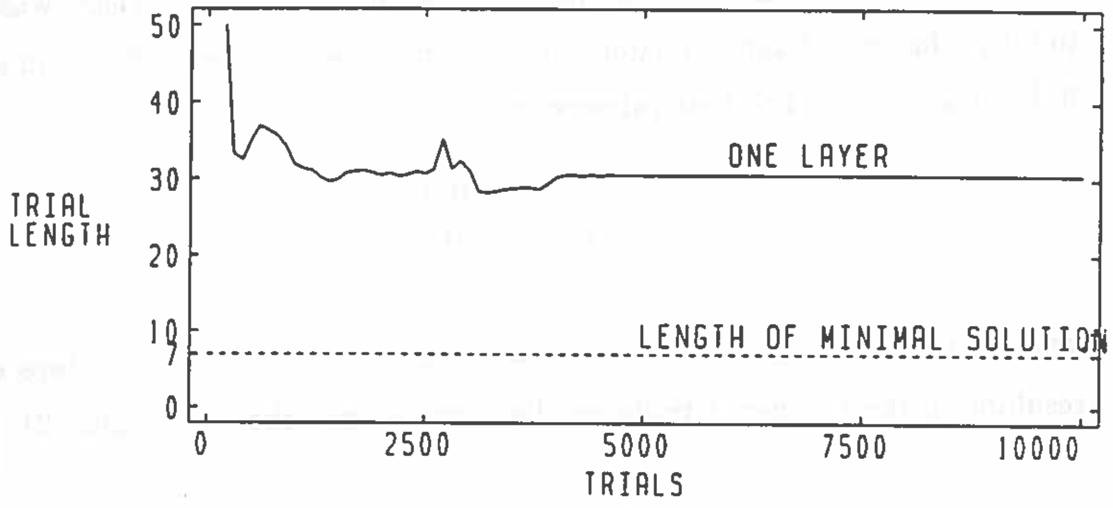

Figure 4: Length of Solution Path versus Trials for One-Layer System

The trial length versus the number of trials is plotted in Figure 4, showing how the length of the solution path varies with experience. The horizontal dotted line in the figure is at a trial length of seven, the length of the minimal solution path. The values plotted are averages over the 10 runs and over bins of 100 trials. The length of the solution path decreases quickly from an average of 50 steps to approximately 30 steps, but performance is never much better than 30 steps per solution. A non-learning strategy of random action selection was found to result in an average of 140 steps per solution path, so the one-layer system significantly improves the initial random search strategy. Note that all runs were terminated before 5,000 trials elapsed—the learning curve was extended as was done for the pole-balancing experiments. The curve might have continued to decrease slightly if the one-layer experiments had been run longer.

The weights learned by the end of Run 7 are shown in Figure 5. The evaluation network has acquired only three weights of significant magnitude, and they are all associated with the encoding of Disk C's peg. The weights' signs result in high evaluations for states with Disk C on Peg 3, the goal peg, and low evaluations for other states. The weights of the action network are more difficult to interpret. Let us postpone the discussion of these weights until the results of the subsequent experiments with a two-layer evaluation network are presented.

We can visualize the learned evaluation function by viewing the state transition graph and at each state (i.e., node in the graph) drawing a circle with radius proportional to the state's

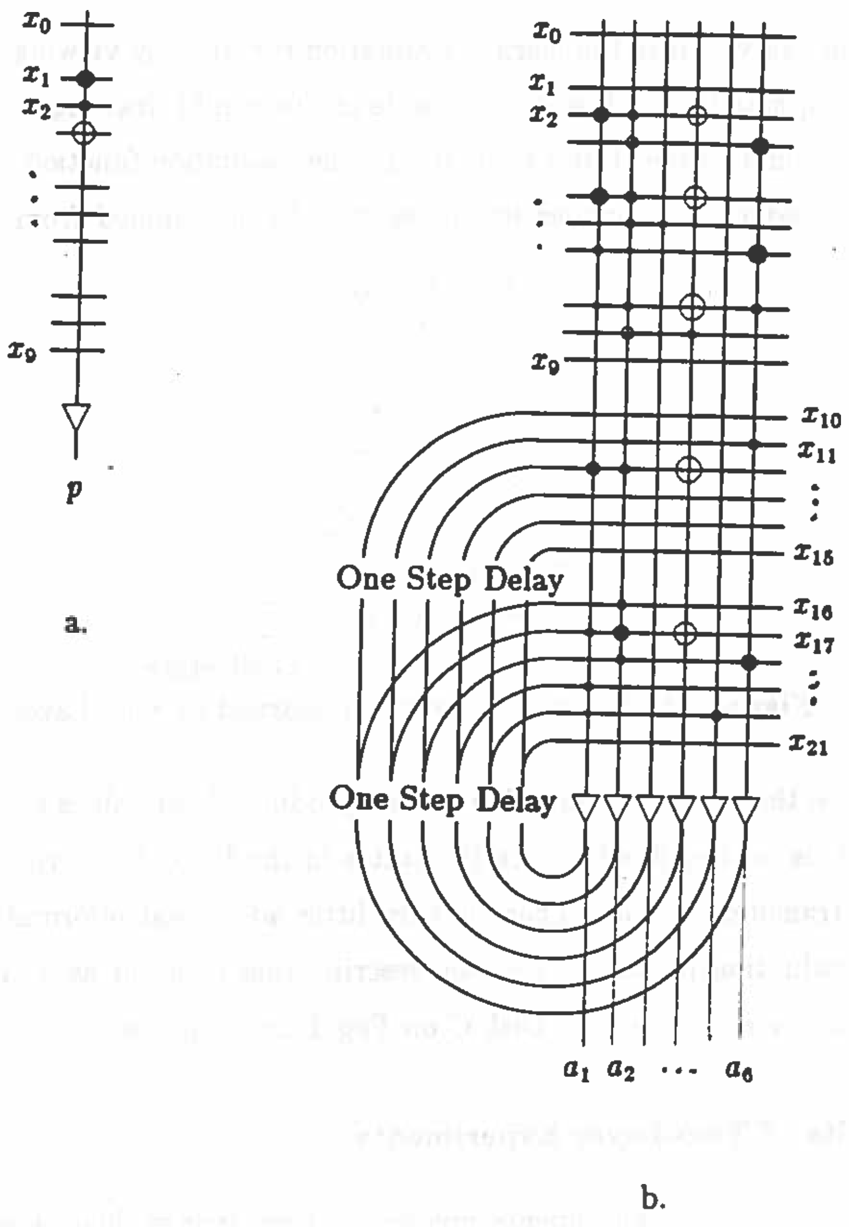

Figure 5: Weights Learned by One-Layer Network

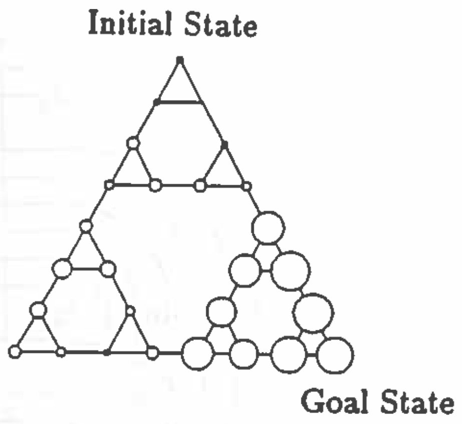

Figure 6: Evaluation Function Learned by One-Layer Network

Run	Trials	Last Trial
1	11,809	7
2	11,418	22
3	11,584	7
4	11,559	7
5	11,967	7
6	12,093	7
7	11,856	7
8	11,636	7
9	12,432	7
10	12,041	7

Table 3: Results of Two-Layer System

evaluation. The evaluation function learned in Run 7 is pictured in this manner in Figure 6. As determined from the signs of the weights, the evaluation function indeed produces high values for states for which Disk C is on Peg 3, which are the states in the large, lower right triangle of the state transition graph. There is very little additional information provided by this evaluation function. We can describe this function as a credit-assignment heuristic, viz., states with Disk C on Peg 3 are desirable.

2.5 Results of Two-Layer Experiments

Our two-layer experiments involved a two-layer evaluation network with 10 hidden units, but with the same one-layer action network as above. We suspected that with the delayed actions as input terms, the one-layer action network could find weight values that result in following the minimal solution path. This is verified by the results of the experiments.

Approximately 20 sets of parameter values were tested by performing 5 runs of 50,000 steps each. The values resulting in the best performance are $\beta = 0.1$, $\beta_h = 2.0$, $\beta_m = 0.9$, $\rho = 0.02$, $\gamma = 0.9$. Momentum was discovered to facilitate learning in this case, though interestingly it retarded learning for the pole-balancing task. Applying these values in 10 runs of 100,000 steps produced the results in Table 3. For all but one run the length of the last trial was seven steps, equal to the length of the minimal solution path. The average number of trials is 11,839, roughly 10 steps per trial averaged over the 100,000 steps. So in 9 out of 10 runs the minimal solution path was learned, and judging from the number of trials in Run 2, the minimal solution path was probably reliably followed in that run, also. There is always a nonzero probability of

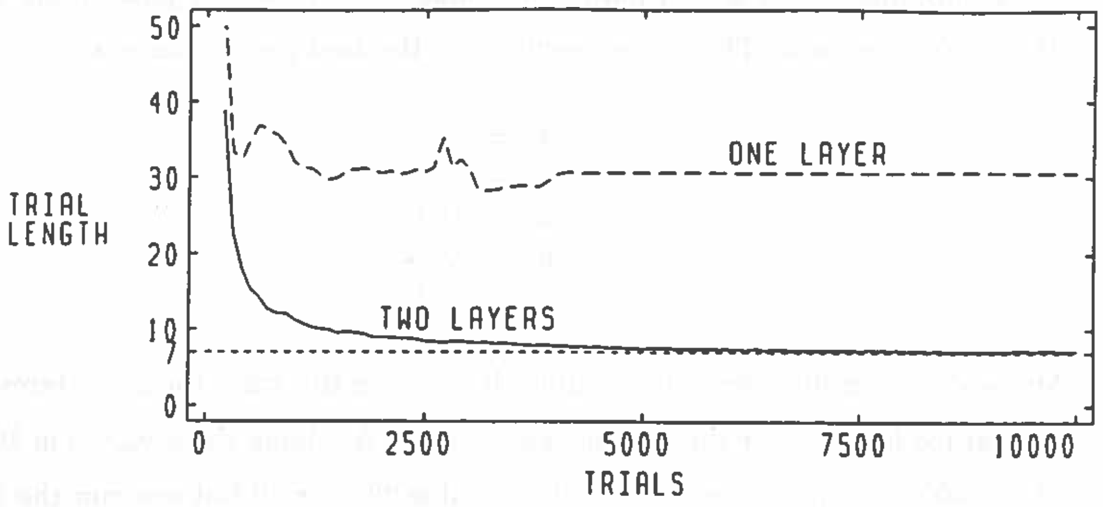

Figure 7: Length of Solution Path versus Trials for Two-Layer System

trying an alternative path, which could explain the last trial of Run 2.

The learning curve of Figure 7 shows that the two-layer system quickly learned solution paths averaging about 15 steps in length, and gradually reduced this to the minimum of seven steps. The learning curve for the one-layer system is superimposed on this graph to highlight the performance increase resulting from the hidden units in the evaluation network.

The weights learned during Run 1 are displayed in Figure 8. First we focus on the weights of the evaluation network. The hidden-unit weights are more varied than they were for the pole-balancing task. Most of the units appear to have acquired useful new features. Units 1, 2, 5, 6, and 9 have weights of large magnitude, though they are by no means the only units of significance. As is usually the case, it is difficult to comprehend what role the units play by studying individual weight values. However, by encoding their output values by the size of circles on the state transition graph, as done earlier for the evaluation function itself, we can learn exactly what the new features are and can gain some intuitions about their contributions to the overall evaluation function. First, we analyze the evaluation function that was learned in Run 1.

2.6 The Evaluation Function and New Features

The value of the evaluation function learned in Run 1 is represented in Figure 9. In comparing two states, the state with the larger circle would be evaluated as being more desirable. Notice that a consistent progression from small circles to larger circles results as one moves from any state toward the goal state by the shortest route, thus this evaluation function is extremely informative. Any search strategy, in addition to the probabilistic method used to generate actions, would benefit from this evaluation function. The sample of the interaction between the learning system and the Tower of Hanoi puzzle shown in Table 1 is actually a trial from the end of Run 1, and one column of that table is the output value of the evaluation network. The value steadily changes from a negative value to increasing positive values as the goal state is approached.

Now let us see how this evaluation function is constructed. Figure 10 shows the output functions for the 10 hidden units, i.e., the features acquired during learning. The radii of the circles for a feature are calculated by scaling the 27 output values for the corresponding hidden unit to be between 0 and a maximum radius. So circles of extreme size do not necessarily

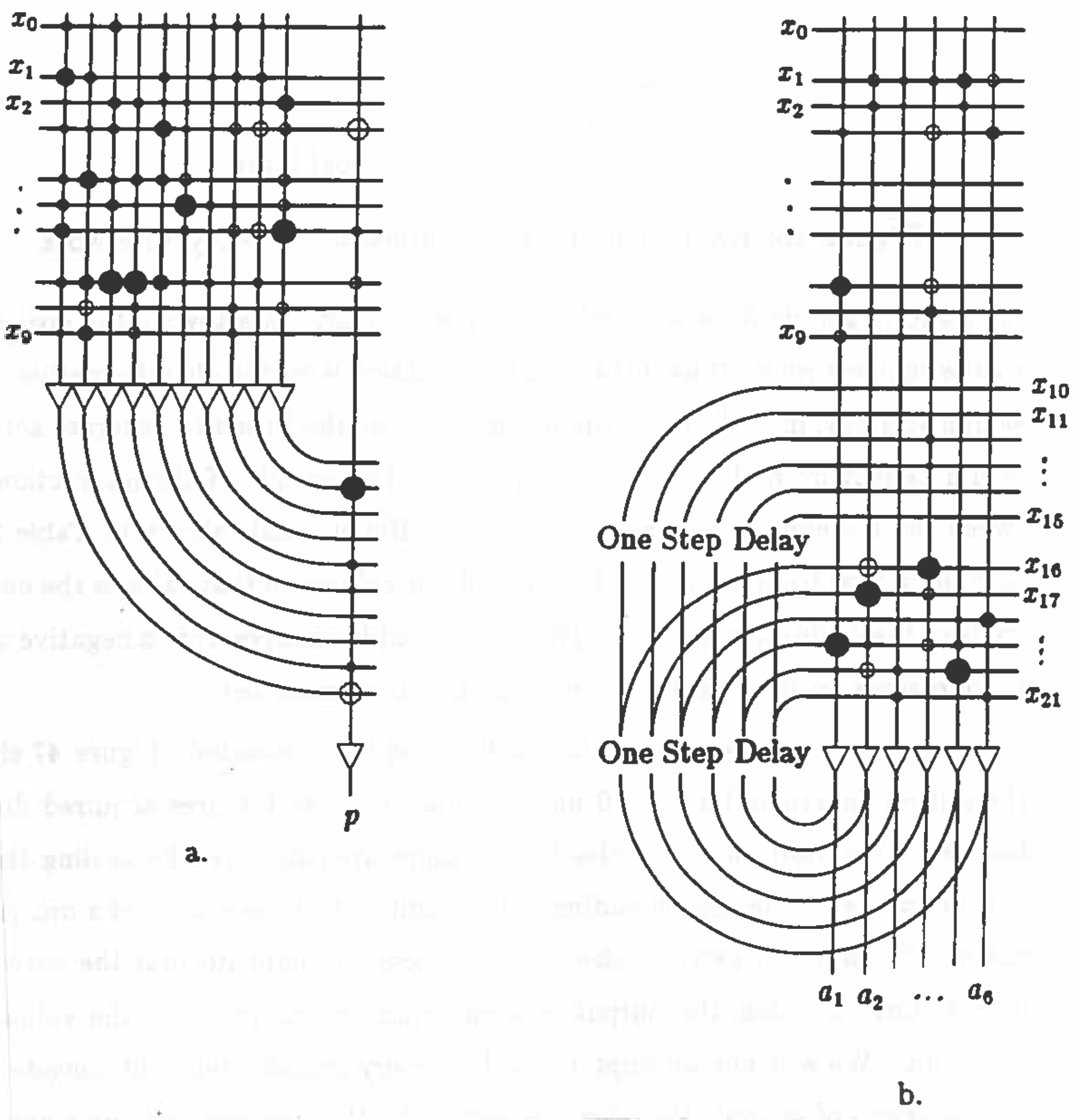

Figure 8: Weights Learned by Two-Layer Network

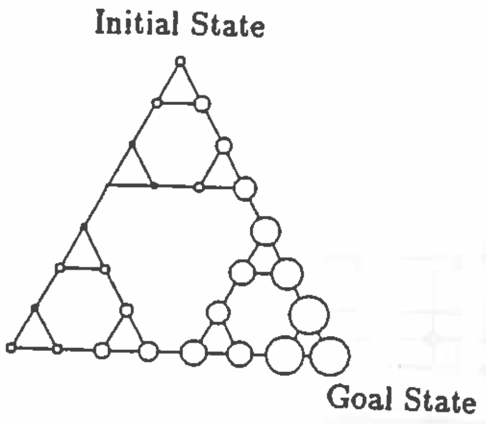

Figure 9: Evaluation Function Learned by Two-Layer Network

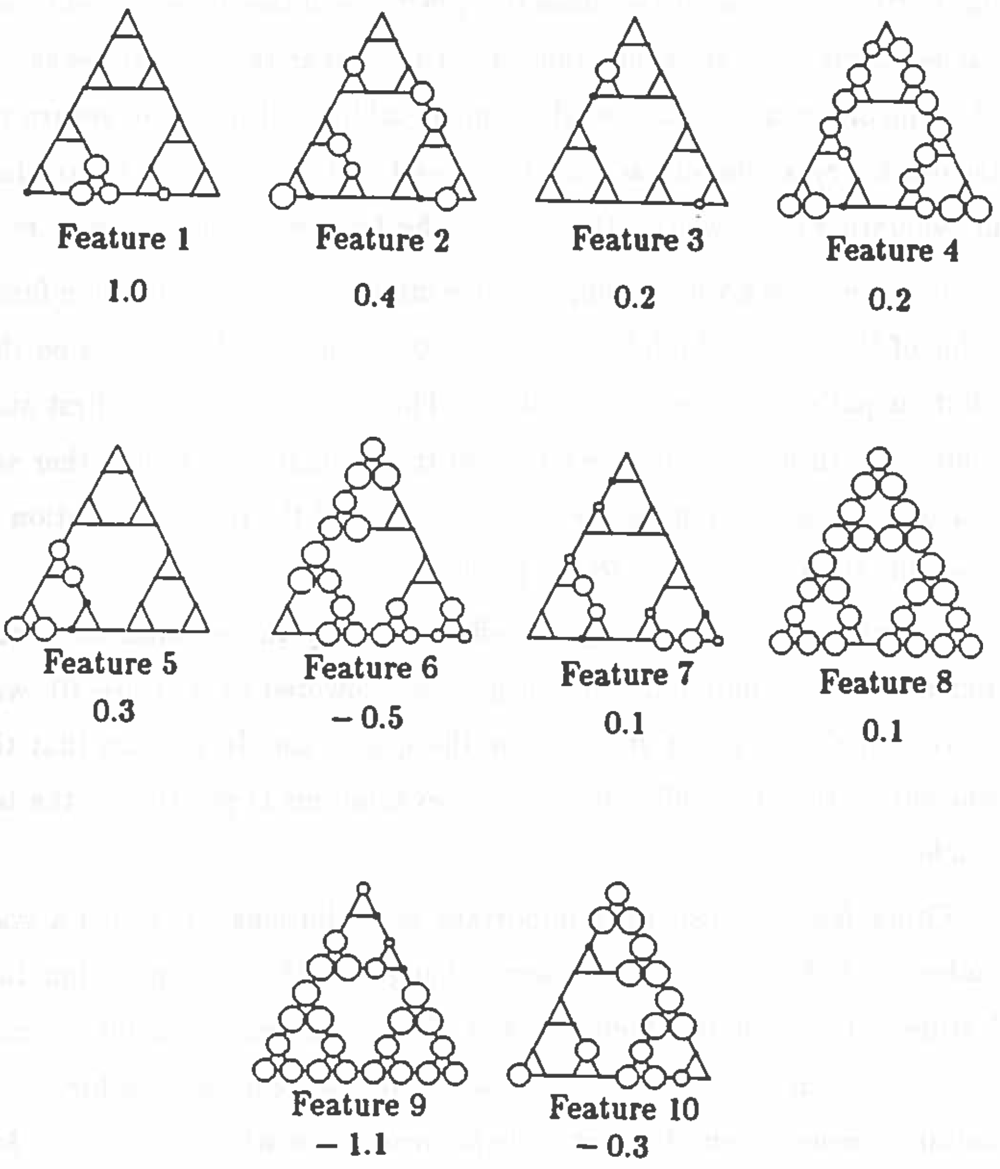

Figure 10: New Features Learned by Two-Layer Evaluation Network

indicate that the output is 0 or 1, but only that the output is a minimum or maximum of the values for that unit. We will not attempt to explain every feature, but will consider the contributions of several. We refer to a feature by the corresponding unit number, such as Feature 1 for the function learned by Unit 1.

Feature 1 has a positive effect on the evaluation. Figure 10 shows that Feature 1 roughly represents three states near the bottom of the graph just outside of the lower right triangular region where Disk C is on the goal peg. Feature 1 boosts the evaluation of these states, thus directing a search from states in the lower left part of the graph toward the state through which the search must pass to get Disk 3 onto the goal peg. This part of the graph is a "bottleneck", and similar bottlenecks exist at the other two junctions of the three largest triangles. The values of the evaluation function are critical near these bottlenecks—the choice of an incorrect action can result in many additional moves to return to the bottleneck to try a different action. Features 1 and 2 seem to be particularly helpful in evaluating the lower bottleneck and the bottleneck on the right, respectively.

Feature 9 has a very strong negative influence on the evaluation function. The value of Feature 9 is high for all states except the first four states on the minimal solution path, plus one nearby state. The evaluations of the first states in the solution path are raised in relation to the evaluations of the other states, thus Feature 9's role is to make the first few states of the minimal solution path more desirable than states next to the path.

Feature 10 also has a negative effect. Mainly the evaluations of states along and next to the minimal solution path are lowered by Feature 10, with the exception of the very last state before the goal state. It appears that this feature guarantees that the difference in state evaluations is positive as the last state is reached.

Other features also have important contributions. Perhaps a good way to understand their roles is to observe changes in the evaluation function as each feature is removed and then restored. From the small amount of analysis done here, it is clear that a variety of new features were developed for this task. The initial representation of the state is far from ideal when it comes to forming the evaluation function, but the combination of the error back-propagation algorithm (Rumelhart, Hinton, and Williams, 1986) with Sutton's adaptive-heuristic-critic algorithm (Sutton, 1984, 1988; Barto, Sutton, and Anderson, 1983) successfully learned sufficient new features for the state representation.

2.7 The Action Function

When the two-layer evaluation network was used, the single-layer action network was able to constrain the search to the minimal solution path. Figure 8b shows that this was accomplished mainly through the development of weight values for the previous-step's action and for the current state. The two-step delayed action did not acquire a significant effect on the selection of the current action.

The large negative weights on the previous action components stand out. Note that there is one large negative weight for each component. These weights have the same effect as did Langley's heuristic for preventing the application of the inverse of the previous action. By tracing the delayed output of each unit to the corresponding negative weight, we find that the negative weights are on the intersections of actions and their inverses. In other words, Action 1-2, or $a_1[t-1]$, has a negative connection to Action 2-1, or $a_3[t]$, Action 1-3 is negatively connected to 3-1, etc. A negative weight lowers the probability that the corresponding action will be generated, and these weights are of such high magnitudes that the probability of the previous-action's inverse is effectively zero.

There are other weights associated with the previous-action inputs that are positive-valued. Through these weights, the generation of an action on one step results in a high probability for a particular action on the next step, thus forming two-step *sequences*. For example, Action 3-2 will be followed by Action 1-3. Referring back to Figure 2 on page 2, this two-step sequence can change the puzzle from the third state on the minimal solution path to the fifth state. Other sequences exist for other two-step transitions along the minimal solution path, and for moving onto the minimal solution path.

The weights on the delayed action components are not sufficient in themselves for limiting actions to movement along the minimal solution path. The current state must at least play a role in selecting the very first action, and indeed it does. Consider the values of the input terms when in the start state (111). All of the delayed-output terms are 0, since this is the first step in the trial. All other input terms are 0, except for the first term of each of the three triples encoding the state. The first of these is connected positively to Unit 2 and negatively to the rest, except for Unit 6, whose action is not legal for the start state. The other two non-zero input terms have small or negative connections to units having legal actions, so Unit 2 will be the unit to respond to the start state. Unit 2 represents Action 1-3, the first action along the minimal solution path. Langley's system was not required to learn the correct action for the initial state, because both states (333) and (222) were goal states—two minimal solution paths exist, and both actions from the initial state (111) move along one of the paths.

3 Transfer of Learning

It is desirable for a learning system to be able to improve its performance on a single task, called *improvement*, and to also improve performance over a set of tasks, called *transfer* (distinction by Ohlsson, 1982). Langley (1985) lists the following four kinds of transfer between tasks:

1. Transfer to more complex versions of the task.
2. Transfer to different initial states or goal states.
3. Transfer to tasks of similar complexity with different state-space structures.
4. Transfer to tasks of little similarity, perhaps requiring some of the same actions (referred to as *learning by analogy*).

The ability of the connectionist learning system to perform the first two kinds of transfer are discussed below.

Langley showed that the heuristics learned by SAGE for solving the three-disk Tower of Hanoi puzzle were directly applicable to the four-disk and the five-disk versions of this puzzle, solving these more complex puzzles with no additional search. The representation of the rules' conditions and actions made this possible: disk, peg, and action names are generalized to variables, therefore the rules can be applied to the new task having an additional disk, since its name can be bound to a variable as well as any previous disk name. In addition, the concept of an action's inverse in included in the system's representation, enabling a situation where an action is followed by its inverse to result in a rule discouraging the use of the inverse of *any* action.

The connectionist representation used here does not permit such generalizations, though learning is transferred to Tower of Hanoi puzzles having more disks. The state transition graph of the four-disk puzzle can be generated by duplicating the three-disk puzzle's graph three times, making an upper, lower left, and lower right major triangle to be connected in a structure identical to that for the three-disk graph, as shown in Figure 11. If the input representation of the networks is augmented by simply adding three components to encode the position of the new fourth disk, assumed to be the largest disk for this discussion, then the action function resulting in the minimal solution path learned for the three-disk puzzle is left intact, as highlighted in the figure. Due to the recursive nature of the Tower of Hanoi family of

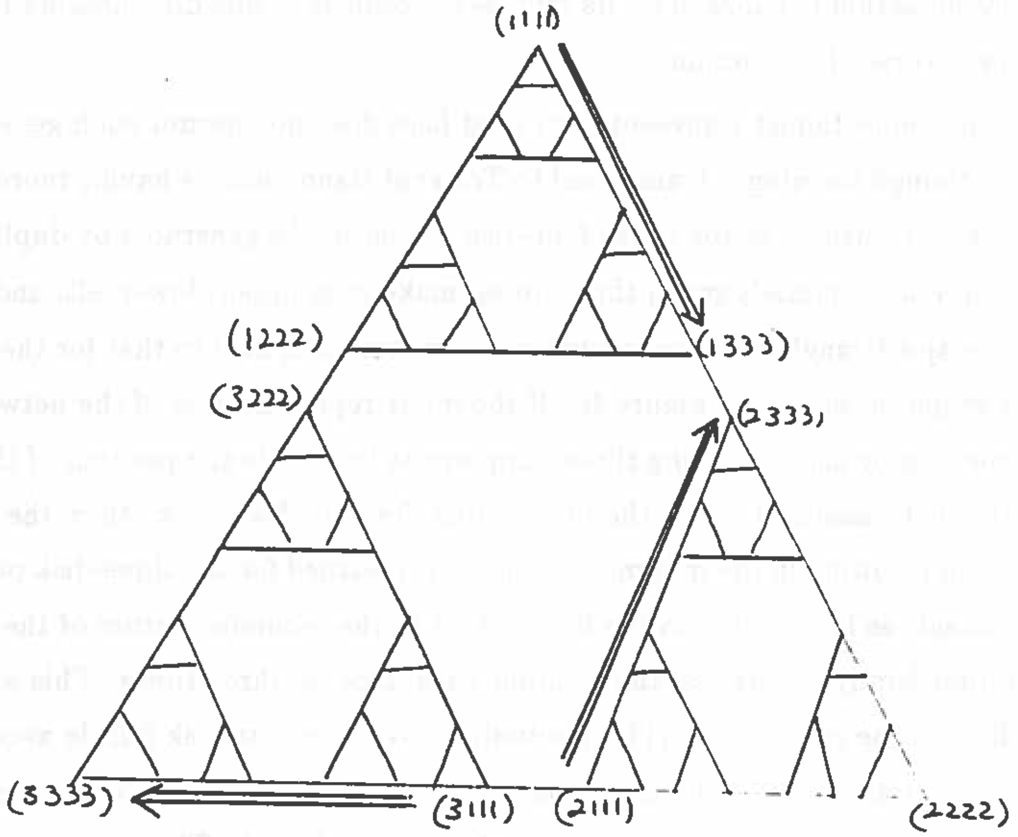

Figure 11: State Transition Graph for Four-Disk Tower of Hanoi Puzzle

puzzles, the solution path appears three times. This solution path from the initial state (1111) actually moves the four-disk puzzle away from the goal state (3333). The action associated with the initial state is wrong, but there are associations that are appropriately transferred. The negative weights preventing the selection of an action that is the inverse of the previous action are still very helpful for the four-disk puzzle. Some of the two-step sequences might also be applicable. To learn the four-disk solution, the evaluation function must also be adjusted, since it is very tailored to the three-disk version. Therefore, the solution of the four-disk puzzle would require additional learning, though probably less than would be needed by a naive system that has no experience with the three-disk puzzle.

The second form of transfer concerns different initial and goal states. Langley's system was not capable of transferring to Tower of Hanoi puzzles with different initial and goal states, but he has shown on another task how the inclusion in the system of a representation of the goal can lead to strategies that are goal-dependent.

The action function learned by our system might generalize correctly to different initial states, particularly those close to the minimal solution path, but this was not tested. The evaluation function does generalize correctly to different initial states. As shown in Figure 9, the evaluations increase for states closer to goal, whether or not the states are on the minimal solution path if the new goal is not close to the original goal. Therefore, learning would be facilitated if the initial state were changed from its original position after the evaluation function had been learned.

As for different goal states, both action and evaluation networks have learned inappropriate functions. In fact, generalization to a puzzle with a different goal state would retard the learning of a new solution path. As Langley suggested, to learn evaluation and action functions for different goals, some representation of the goal must be included as input to the networks. This could be done very simply by duplicating the terms of the current state representation and using them to encode the goal state. Different evaluation and action functions would then be learned for different goals, though a multilayer action network would probably be required.

4 Conclusion

Anderson (1987) demonstrated connectionist learning algorithms solving a task having a *large search space*, delayed reinforcement, and requiring non-trivial (nonlinear) combinations of features. Here, its generality was tested by applying it to a task with a *small search space*, requiring non-trivial feature combinations, and for which reinforcement is delayed *and* infrequent. We have shown how some of the credit-assignment techniques that have been developed for learning rules *while doing* can be incorporated into a reinforcement scheme.

The connectionist learning system was able to learn the solution to the three-disk Tower of Hanoi puzzle. The time (amount of experience) required to solve it is much greater than that required by Langley's (1985) adaptive production system, but fewer assumptions are made by the connectionist implementation. A very limited input representation was used, consisting only of the current state and the two-previous actions, though this is sufficient for a solution to be learned.

The speed of learning could be accelerated in a number of ways. The learning algorithms are just a first attempt at combining reinforcement-learning techniques with a particular method for learning in multilayer networks. The combination of reinforcement-learning with other multilayer learning algorithms should be investigated. Some multilayer learning algorithms are more appropriate when credit-assignment methods can be completely trusted, and can remove errors in a single step (such as Reilly, Cooper, and Elbaum, 1982). Such approaches can lead to a proliferation of features—in the limit one for every state. This would not be a problem for the three-disk Tower of Hanoi task, but it is not a general solution for tasks with a large number of states.

Comparisons of this connectionist approach for strategy learning with symbolic approaches highlights some of the limitations of connectionist representations. For example, the connectionist system used for the Tower of Hanoi experiments is not capable of doing variable binding in the way that Langley's (1985) production system can. Langley's system was able to learn a single symbolic rule that uses action variables to prohibit actions that are the inverses of the previous actions. Langley's production system was able to learn such rules using built-in knowledge of what "inverse" means and how particular actions and states can be generalized to variables. This raises two questions of fundamental importance to the research of connectionist systems. First, how can variables be represented in connectionist representations? This is a well-known limitation of connectionist representations (e.g., Touretzky and Hinton, 1985). In our implementation, actions are not generalized to variables; distinct negative weights from each action to its inverse had to be learned. Second, given that variables are not represented, how can task knowledge, such as the "inverse" action concept, be used to generalize one experience to other actions? For the action representation used in this chapter, this would entail some mechanism that a) observes the negative change in the weight between the previous action and its inverse, and b) uses knowledge of which weights connect other actions and their inverses to duplicate this weight change for the other actions. The answers to both questions will certainly involve connectionist structures of greater complexity than the small networks used in this study. Perhaps a connectionist implementation of symbolic processing is required (e.g., Touretzky, 1986).

5 References

Amarel, S. (1981) Problems of Representation in Heuristic Problem Solving: Related Issues in the Development of Expert Systems. *Technical Report CBM-TR-118*, Laboratory for Computer Science Research, Rutgers University, New Brunswick, NJ.

Anderson, C.W. (1987), *Strategy Learning with Multilayer Connectionist Representations* Technical Report TR87-509.3, GTE Laboratories, Incorporated, Waltham, MA, 1987. (This is a corrected version of report published in *Proc. Fourth International Workshop on Machine Learning*, Irvine, CA, pp. 103–114, June 1987.)

Anzai, Y. and Simon, H. A. (1979) The Theory of Learning by Doing. *Psychological Review, 86.*

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983) Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics, 13,* 835–846.

Langley, P. (1985) Learning to search: from weak methods to domain-specific heuristics. *Cognitive Science*, *9*, 217–260.

Luger, G. F. (1976) The use of the state space to record the behavioral effects of subproblems and symmetries in the Tower of Hanoi problem. *International Journal of Man-Machine Studies*, *8*, 441–421.

Nilsson, N. J. (1971) *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, Inc.

Ohlsson, S. (1982) On the automated learning of problem solving rules. *Proceedings of the Sixth European Meeting on Cybernetics and Systems Research*, 151–157.

Reilly, D. L., Cooper, L. N. and Elbaum, C. (1982) A neural model for category learning. *Biological Cybernetics*, *45*, 35–41.

Rumelhart, D. E., Hinton, G. E., and Williams, R. (1986) Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. by Rumelhart, D. E., McClelland, J. L., and the PDP research group., Cambridge, MA: Bradford Books.

Sutton, R. S. (1984) Temporal aspects of credit assignment in reinforcement learning. University of Massachusetts Ph. D. Dissertation.

Sutton, R. S. (1988) Learning to Predict by the Methods of Temporal Differences, *Machine Learning*, vol. 3, pp. 9–44, 1988.

Touretzky, D. S. (1986) BoltzCONS: reconciling connectionism with the recursive nature of stacks and trees. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA., pp. 522–530.

Touretzky, D. S. and Hinton, G. E. (1985) Symbols among the neurons: details of a connectionist inference architecture. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA.