CHAPTER V

EMERGENCE OF TASK-DIRECTED COMPONENT MANIPULATION

5.0 Emergence of Task-Directed Component Manipulation

In the previous chapter, the natural dynamics of an evolutionary weak method were used to induce both the architecture and parametric values for a neural network. The demonstrated dynamics were a natural component of the evolutionary algorithm used.

In this chapter, representation specific operators are added to an evolutionary weak method, similar to GNARL from the last chapter, that designate which components of a finite state machine (FSM) representation should and should not be manipulated by the other operators in future generations. As with other operators in evolutionary weak methods, these operators are applied randomly. Because the operators are representation specific and they are applied randomly, the knowledge that determines those components that should be protected emerges from the interaction of the evolutionary weak method with the task.

Two effects result from the operators introduced in this chapter. First, those components that are crucial to solving a task, i.e., task-specific structures, are identified and protected from future manipulation. The knowledge that determines which of components should be protected emerges during problem solving. Second, as a result of identifying those components that are task-specific the time to acquire FSMs for the demonstrated task decreases. This demonstrates not only that interaction with the task can direct which features should be manipulated but that doing so is beneficial to the acquisition process.

5.1 Evolving Finite State Machines

The program used in this chapter is similar to the GNARL program of last chapter except the representation being evolved is not neural networks but FSMs. Finite state machines were the original representation used in early evolutionary programming (Fogel,

Owens and Walsh 1966). More recently Tomita (1982) uses an evolutionary program to induce FSMs for languages and Fogel (1992a; 1992c) describe evolving FSMs for various experiments on the Prisoner's Dilemma (Luce and Raiffa 1957; Rapoport 1966); Axelrod 1984).

Accordingly, the representation specific operators in this program manipulate various representational aspects of an FSM. The components of an FSM are states and transitions. Each transition is associated with an input symbol and an output symbol. This suggests separate operators to add states, delete states, add transitions, delete transitions and modify transitions.

The method of evolving FSMs used in this chapter is slightly different than the methods described in Fogel, Owens and Walsh (1966) and Fogel (1992a). First, during early experiments with this problem, the algorithm created individuals with larger and larger numbers of states until reaching the allowed maximum. In addition, a disproportionate percentage of the population acquired the same fitness for a considerable amount of time. In these early experiments, there was an equal chance for adding a state, deleting a state or modifying a transition.

The number of mutations made to a parent to create an offspring in this program is given by the function:

$$1 + \text{round}(\text{abs}\,[N(0, T) \times size]) \qquad \text{(EQ 22)}$$

where *size* is the number of transitions in the parent FSM and $N(0, T)$ is a gaussian random variable with mean 0 and variance proportional to the fitness "temperature" of the parent. The method of selection was the competitive method described in Fogel (1992a) and outlined in Chapter 2 with the exception that if the two population members being compared had the same fitness, a "winner" was chosen randomly. The number of competitions per individual was 5. The population was sorted by their competitive selection scores with the best half of the population retained and replicated to create the following generation. Each population member created exactly one offspring for the next generation. A population size of 300 machines was used for each run.

After analyzing a few evolved FSMs with the above algorithm it became apparent that a large percentage of the states in the resulting machines were in fact unused. Because of

the high probability of deleting a state, the additional states ensure a high percentage of self-replication for the FSMs. By including a large number of superfluous states, whenever a state deletion is performed, there is a better chance that the offspring will retain the ability of its parent. This accounts for the inordinate number of machines with the same fitness and maximum number of states. It is interesting to note that the dynamics of the evolutionary weak method allowed this safeguard to emerge without being anticipated by the programmer.

To discourage the introduction of unnecessary states, the probability of applying the operators for altering an FSM were modified so that there was an even chance of mutating either a state or a transition. If a state mutation is selected, the chance of deleting a state as opposed to adding a state is given by:

$$P(delete) = \frac{numstates}{maxstates} \qquad \text{(EQ 23)}$$

where *numstates* is the number of states in the parent FSM and *maxstates* is the maximum number of states allowed for the problem. Thus if the number of states in the parent is less than half that allowed for the problem, there is a greater chance of adding a state than deleting a state. When the number of states in the machine is more than half of the total number allowed, there is a preference for deleting states. While this did not entirely curb the tendency for the runs to approach the maximum number of states, it did allow for a consistently broader distribution of sizes in the population and improved the overall acquisition times.

5.2 Freezing and Unfreezing Representational Components

To identify task-specific component manipulations in the evolving individuals, two operators are added to the basic program described above. The first operator, called *freeze,* selects a random portion of the FSM and signifies in the representation that it is to be preserved from future modification. The second operator, *unfreeze*, is the opposite of freeze.- Unfreeze releases a frozen representational component so it can once again be manipulated by the reproduction operators. The opposite actions of the freeze and unfreeze operators is important to allow the set of features being preserved to be adaptable to the changing dynamics of the task.

Because there is no single, general method of identifying what portions of the individual should be preserved, the components modified by these operators are selected at random. By the arguments of the Chapter 3, if a randomly selected freeze operation protects crucial components of the representation from modification, thus posing a benefit to the reproductive ability of the individual, then this "feature" is passed on to its offspring. Likewise, if a frozen component is detrimental to the member's proliferation, i.e., it protects an erroneous component, then that modification is eventually removed from the population. Referring back to Equation 8, the protected components are the feature of interest, μ, and $\varepsilon(\mu, t) = 0$ since the reproductive operators cannot modify the protected component. Equation 8 then shows that $\overline{\sigma}(\mu, t) > 1/n$, where n is the size of the population, is a sufficient average selection probability to propagate the protected component through the population.

5.3 Experiments

To illustrate the effects of freezing and unfreezing, the ant problem, described in the previous chapter, was again used, but more like the form of Jefferson et al. (1992) than in chapter 4. In this experiment, the goal is to evolve an FSM controller to guide the artificial ant along the path of food shown in Figure 17 within 200 time steps. As before, the ant is equipped with a single sensor that can detect the presence or absence of food in the square directly in front of it. Actuation of the ant is signaled through four possible action commands: move one square forward (MOVE), spin left 90^o (LEFT), spin right 90^o (RIGHT), or do nothing (NOOP). On each time step, the ant executes an implicit sense/act loop where an input of FOOD or NOFOOD is given to the ant and it executes a single action command. Once the ant enters a position on the grid with food, the food is removed and a point of fitness is awarded.

As previously stated, Jefferson et al. (1992) used a genetic algorithm to compare the evolution of bit strings that were interpreted as either finite state machines (FSMs) or recurrent neural networks depending on the experiment. Jefferson et. al. (1992) used a population size of 65,536 and replaced 95% of the population each generation for both representations in both experiments. Evolving an FSM controller for this problem took 52 generations while the neural network controller took 94 generations to emerge. Thus their genetic algorithm searched a total of 3,303,014 FSMs and 5,917,900 recurrent neural networks to solve the ant problem.

In the following experiments, FSM controllers for the ant problem evolve using evolutionary programming with and without the freezing operators. A single freeze operation selects a single state and up to 5 transitions at random and designates them as being "frozen" in the representation. Conversely, unfreeze releases a single randomly selected frozen state and up to 5 frozen links. When creating an offspring there is a 10% chance of applying a freeze operation and then a 20% chance of applying an unfreeze operation. The higher unfreeze rate is to ensure that if local minima are reached the number of protected components will decrease and allow protected components to be mutable again. All freezing and unfreezing operations are applied to the offspring prior to the other mutations. At most 75% of the states and 75% of the transitions for any one FSM are permitted to be frozen at a time.

In order to provide slightly more discriminations between evolved FSMs, the fitness function for these experiments modified is:

$$food + 0.01\ (1 - \frac{t}{200}) \qquad \text{(EQ 24)}$$

where *food* is the amount of food found by the ant within 200 time steps and t is the time step on which the last piece of food was found. This fitness function encodes a preference for FSMs that acquire the same amount of food in fewer time steps.

To determine the effect of the freeze and unfreeze operators on the evolution of FSM controllers for the ant problem, eight runs were executed, four with compression and four without.

5.4 Results and Discussion

Table 6 shows the total number of FSMs constructed in each run until one FSM in the population guided the ant to all 89 pieces of food within the allotted 200 time steps. The number of generations for each run is listed in the parentheses. The runs are sorted into increasing order so that the fastest and slowest of both methods are compared directly. Speed-up, shown in the last column, was computed by dividing the result from the run without freezing by the result of the run with freezing. Two additional runs, one with freezing and one without, did not find a solution within the maximum 5000 generations and have not been included in Table 6.

Table 6. *Speed-up results for FSA induction for 4 runs with and without freeze and unfreeze.*

Run	*Number of FSMs Evaluated (Number of Generations)*		*Speed Up*
	Without Freeze/Unfreeze	*With Freeze/Unfreeze*	
1	187,950 (1251)	63,000 (418)	2.99
2	269,850 (1797)	152,100 (1012)	1.77
3	331,200 (2206)	156,600 (1042)	2.12
4	734,850 (4897)	607,950 (4051)	1.20

First notice that all of the reported runs, both with and without freezing, evolved solutions to the ant problem more quickly than Jefferson et al. (1992) which constructed 3,303,014 FSMs. The improvements range between factors of 52 and 4.5. Such improvement is often the case when converting from a genetic algorithm using binary representations to an evolutionary program. This is because the added complication of an interpretation function to convert between the genotypic and phenotypic representations in the genetic algorithm is avoided in evolutionary programming. Whether or not the differences between the two evolutionary algorithms is solely responsible for the improved results or if our modified fitness function also contributed to the speed-up is unknown. Regardless, the improvement over Jefferson et al. (1992) both in the previous chapter using recurrent neural networks and here with FSMs is noteworthy.

Next, notice that each of the runs show speed-up in favor of those experiments that use the added freeze operators. An explanation for these results is that the freezing process identifies and protects components that are important to the viability of the offspring. Subsequent mutations are forced to alter only less crucial components in the representation. This removes from consideration a number of non-viable modifications that are routinely explored when freezing is unavailable. The fact that no knowledge of the task is used to

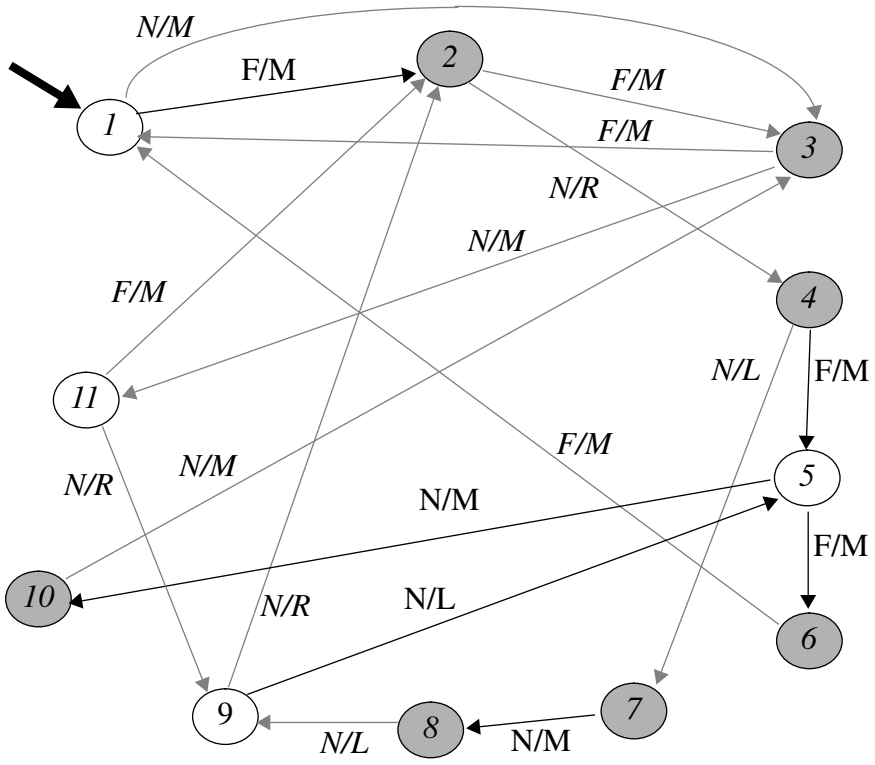

Figure 23: FSM evolved with compression in 420 generations. Frozen states and transitions are shown in grey. Input symbol set is (F, N) and output symbol set is (M, L, R, N) as described in the text. Extraneous states and transitions are not shown. The initial state is indicated by the oversized arrow.

determine which components to freeze indicates the presence of emergent knowledge. By interacting with the task environment, components of the FSMs that are necessary to the task are identified through the behaviorial results after they are modified.

Figure 23 shows the evolved FSM from run 1 with freezing. Frozen transitions and states are shown in grey. This FSM guides the ant to all 89 pieces of food in 193 time steps. There are a total of 22 states in this solution only eleven of which are used to solve the problem. Twelve of the states and 23 of the 44 possible transitions are frozen in this solution. It is difficult to deduce any easily described significance for the frozen components in this FSM. The non-determinism of the operator places an emphasis on whatever gets results. One observation is that states 1 through 6 and the transitions between them are largely protected from modification. This portion of the FSM is responsible for tra-

versing the continuous stretches of food of the path. Mutation of one of these states or transitions has a high probability of creating an offspring far below optimal.

In a population, different members of the population generally encode vastly different approaches to solving the task. This means that a standard approach to credit assignment would need to contain knowledge about each of the various possible solutions in order to determine which components of each FSM to modify. This amount of explicit knowledge would be prohibitive and would involve accurately determining which solution the FSM approximated most closely. However, empirical credit assignment allows no explicit task-specific knowledge to determine those components that are crucial to the solution and those that should be considered for modification.

5.5 Conclusion

This chapter demonstrates that components of an FSM that are imperative to solving the problem can be identified using emergent techniques. This is exploited as a weak form of local credit assignment to speed-up the FSM acquisition. The experiments demonstrate that emergent techniques can identify task-specific features of a representation without the need for explicit task-specific knowledge.