Evolving Finite-State Machine Strategies for Protecting Resources

William M. Spears and Diana F. Gordon

AI Center, Naval Research Laboratory, Washington DC 20375, USA,
spears@aic.nrl.navy.mil,
WWW home page: http://www.aic.nrl.navy.mil/~spears

Abstract. We are becoming increasingly dependent on large interconnected networks for the control of our resources. One important issue is resource protection strategies in the event of failures and/or attacks. To address this issue we investigated the effectiveness of evolving finite-state machine (FSM) strategies for winning against an adversary in a challenging Competition for Resources simulation. Although preliminary results were promising, unproductive cyclic behavior lowered performance. We then augmented evolution with an algorithm that rapidly detects and removes this cyclic behavior, thereby improving performance dramatically.

1 Introduction

We are becoming increasingly dependent on large interconnected networks for the control of our resources, such as the Internet, communications networks, and power grids. The advantage of these networks is the ability to route resources in a reasonably optimal fashion. However, their interconnectivity, coupled with the lack of global view of what is happening in these networks, can lead to tremendous problems in network reliability. For example, small local failures can easily propagate to entire networks, causing loss of service and corruption of data. Also, deliberate attacks (such as "denial of service" [3] attacks) can easily cause widespread havoc, as poignantly demonstrated recently [11].

Thus one important issue is the development of effective network traversal strategies to protect as many resources as possible from failure and/or attacks, i.e., to maximally restrict the number of resources damaged. To address this issue we have decided to create a "resource protection" simulation that captures the essential aspects of this problem. A "defender" attempts to protect resources before they are damaged by an intentional (or unintentional) "adversary".

Our primary goal then is to create sophisticated reactive strategies for the defender. We use finite-state machines (FSMs) for our strategies, since there are a number of precedents for FSMs being effective strategies for adversarial situations. We use evolutionary algorithms (EAs) to create the FSMs, since there is ample evidence for the effectiveness of this approach [4, 5].[1] This paper serves to summarize and highlight the results we have obtained thus far.

[1] The evolution of FSMs is often referred to as "evolutionary programming" or "EP".

2 The Competition for Resources Problem

Our current Competition for Resources simulation is a novel two-player game on a toroidal board of squares. Each square corresponds to a resource, and the two players (the "defender" and "adversary") compete for squares on the board. If the board is of size $N \times N$, then the defender will start at square (1,1) and the adversary will start at square (N,N). The remaining squares are initially unoccupied. Since the board grid represents real networks, such as power grids or communication networks, and in the real world networks may be highly interconnected and will have few geophysical boundaries, our board is toroidal (has no edges). In this paper we assume $N = 10$, which is quite challenging.

Each player can only perceive limited information, namely, the status of the north, south, east, and west squares neighboring the current position of the player. The diagonal squares can not be seen. The status of each neighboring square will be one of the following: unoccupied, occupied by that player, or occupied by the opponent. Due to the toroidal nature of the board, the defender and the adversary are close to one another at the beginning of the game. However, because they can not see along diagonals, they can't see one another initially.

Each time step the players alternate taking an action, which consists of moving to a neighboring resource to control/protect that resource. A player can move to an unoccupied square or back to a square that it has previously occupied, but not to a square occupied by the opponent. The player isn't allowed to "stand still" and make no move. However, because each player must follow a path of "owned" resources to its current position, it will always be able to make a move at every time step (it can always back up along the path it has taken). Thus a player can not be "trapped" at a square, i.e., it can not be completely surrounded by the opponent. Once an agent occupies a resource, it controls/protects that resource forever. A game ends when all squares are occupied or time runs out. The agent with the most resources at the end of the game wins.

Throughout this paper the adversary will have a fixed stochastic strategy that the defender must "learn" to defeat. The strategy we have chosen for the adversary is simple, but is surprisingly hard to beat. If the adversary detects any unoccupied neighboring squares, it uniformly randomly moves to one of them. Otherwise it uniformly randomly backtracks to a neighboring square it has previously occupied. Given the game and our adversary, we focus on developing effective strategies for the defender.

3 Overview of Finite State Machines

FSMs can be effective representations of agent plans/strategies, e.g., see [1] or [10]. The type of machine used here allows for indeterminate-length action sequences. Recall from Hopcroft and Ullman [9] that the usual acceptance criterion for finite-length strings is termination in a "final" state. Here we assume that there are no final states, i.e., action sequences of any length are allowed. This provides a good model of embedded agents that are continually responsive to their environment.

Formally, we define the machine M to be a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_1)$. Q is the set of vertices (states) of M, Σ is the alphabet of input symbols (which are agent sensory inputs), and Δ is the alphabet of output symbols (which are agent actions). δ is the transition function from a state and an input to a next state, i.e., $\delta(q_i, x_i) = q_{i+1}$ where q_i, $q_{i+1} \in Q$ and $x_i \in \Sigma$ is a sensory input. λ is the transition function from a state and an input to an output, i.e., $\lambda(q_i, x_i) = a_i$ where $q_i \in Q$, $x_i \in \Sigma$, and $a_i \in \Delta$ is an action. The initial state is q_1.

We assume that the FSMs are *deterministic* and *complete*. The FSMs are deterministic because δ and λ are functions, i.e., for every state and input there is a unique next state and action. The FSMs are complete because there exists a next state and action for every state and input, i.e., δ and λ are fully defined. Deterministic and complete FSMs are strategies that tell the agent precisely what to do in every situation it perceives.

For an FSM strategy for the Competition for Resources simulation, the sensory input x_i shows the status of the neighboring resources immediately to the north, east, south, and west of the agent. The status of each resource can be 0 (unoccupied), 1 (occupied by the defensive agent), or 2 (occupied by the adversary). Thus an input of "2100" specifies that the north resource is owned by the adversary, the east resource is owned by the defensive agent, and that the south and west resources are unoccupied.

4 Evolution of Finite State Machines

In an EA a population of P individual structures is initialized and then evolved from generation t to generation $t+1$ by repeated applications of fitness evaluation, selection, recombination, and mutation. In the context of the Competition for Resources simulation, each individual in the population is an FSM. Each FSM is evaluated by playing the game numerous times, to obtain an estimate of how well that FSM is defending the resources against the adversary. Those FSMs that perform the task better are allowed to have more children, which are created through the processes of mutation and recombination. This process continues generation by generation, until termination.

Representation. For efficiency we chose a simple tabular representation for the FSMs. Rows in a table correspond to states, and columns correspond to inputs. For each state q_i and input x_j, table entry (i,j) has two elements. The first element is the next state, i.e., it is $\delta(q_i, x_j)$. The second element is the action to take given the agent is in this state q_i and sees input x_j, i.e., it is $\lambda(q_i, x_j)$.

The number of states S is user defined. The maximum number of inputs is $3^4 = 81$, since the status of each of the four neighboring squares may have three values (0, 1, and 2). However, the input "2222" will never occur, since that implies that the defender is surrounded by the adversary. This is impossible, since the defender must have been able to get to the square it currently occupies. Thus there are 80 possible inputs and we require a table of size $S \times 80$. Each entry in the table is an "allele" that represents a next-state/action pair. Since each allele

is defined uniquely, the FSM is guaranteed to be deterministic and complete. The initial state is always state 1.

Initialization. Throughout this paper a population size of $P = 100$ is assumed, since it produced good results. Each of the P FSMs at generation zero is initialized by using domain-specific knowledge. For any given state (row) and input (column) the next state is chosen uniformly randomly from the set of all S states. However, the choice of action is somewhat more complex. The number of possible actions is maximally four, since the defender may potentially move north, east, south, or west. However, in practice, some of these moves might be impossible, if the adversary owns the neighboring squares.

For example, suppose again that the input is "2100". In this case the north resource is owned by the adversary, and there are only three *legal* moves: east, south, and west. Moving north is *illegal*, since the adversary owns that square. Actions are restricted to those that are legal, and every input has a set of legal moves that are possible. However, since the goal of the game is to capture resources, we also found it useful to define *preferred* moves – those that capture previously unoccupied squares. When the input is "2100", moves to the south or west capture new territory and are thus preferable. During initialization actions are always chosen uniformly randomly from the set of preferred actions, if there are any. If there are no preferred actions, then a legal action is randomly chosen.[2]

Adapting the Number of States. Standard methods for evolving FSMs adapt the number of states [4]. State adaptation raises a number of issues. When a state is deleted, should one really erase the state from the table, or should it simply be made inaccessible? Our prior experience in similar areas has shown that it is often best to make the information inaccessible [2]. Since information has been learned, keeping the information stored serves as a useful memory, which can be re-activated at a later time (if the state is added back to the FSM). Thus we added a "tag bit" to each row of the FSM table. If the tag is 1 the state is accessible. If the tag is 0 the state is inaccessible, but is not destroyed. When a state is added, it is accomplished simply by turning on the tag. Tag bits are subject to an independent mutation operation, that flips the tags with probability 0.001. Since state 1 is always the initial state, it can not be made inaccessible.

Once a state q_j has been made inaccessible, how should the remainder of the FSM (that points to that state) be "repaired"? We investigated two solutions: (1) if state q_i points to q_j, change the pointer so that it points back to q_i and (2) change the pointer to point to any state that is accessible, chosen uniformly randomly. The latter solution performed better.

Adapting FSM Table Entries. Adaptation of the FSM table entries is accomplished with mutation and recombination. Mutation is reasonably straightforward. Each allele (next-state/action pair) in the FSM is chosen with probability p_m. Once an allele is chosen a coin is flipped to see whether the action or the next state is mutated. With probability p the next state is mutated by uniformly randomly choosing a state from the set of all accessible states. With

[2] The emphasis on preferred actions enormously helps the initial search of the EA.

probability $1 - p$ the action is mutated. If there are any preferred actions the algorithm uniformly randomly chooses one of those. If there are no preferred actions the algorithm uniformly chooses any legal action. Note that this could result in no change (e.g., if there is only one legal action). Experiments indicated that performance was remarkably insensitive to p_m and we use $p_m = 0.001/S$ throughout this paper. Setting p to 0.5 worked well.

We also use P_0 uniform recombination [12]. A proportion p_r of pairs of parents in the population are chosen for recombination. For each pair of parents, a coin is flipped for each of the $S \times 80$ alleles. The allele at the table location (i,j) in the first FSM is swapped with the corresponding allele in the second FSM, with probability P_0. If alleles are swapped, both the next state and the action are swapped. Since only corresponding alleles are swapped, there is no need to worry about possible illegal actions. If an action is legal for one FSM at location (i,j) it must be legal for any other FSM at location (i,j), since the input j is the same. Since parents may have different sets of accessible states, recombination may swap alleles in such a fashion that a next state that was accessible in one child FSM is now inaccessible in the other FSM. In this situation a new next state is chosen uniformly randomly from the set of accessible states (in that FSM). Experiments indicated that performance was very sensitive to p_r and P_0. Using recombination to its fullest extent ($p_r = 1.0$ and $P_0 = 0.5$) worked best.

Fitness Evaluation. Since the adversary in the Competition for Resources game is stochastic, each defender FSM will have to play the game multiple times in order to obtain an estimate of how well it defends the resources. Recall that the player with the most resources at the end of the game wins. In case of a tie, the adversary wins. Given G games, the fitness of a defender FSM is the fraction of games that it wins. This fitness function returns values from 0.0 to 1.0, with 1.0 representing an FSM that won all the games it played. Setting G properly proved to be difficult. Prior work [7] concluded that the overall efficiency of the EA may often be improved by reducing G and by running for more generations. This did not work for us. A low value of G resulted in unacceptable sampling error and a high value was too CPU intensive. We were unable to balance these constraints with an intermediate value.

To solve this difficulty we took a two-phase approach. Initially, we use a low value of G, so that each individual can get a quick evaluation. If that individual is promising (it did better than the best individual seen thus far), it is re-evaluated using a high value of G. If it still beats the best individual thus far, it becomes the new best individual. The idea was to carefully evaluate only those individuals that appeared promising. This approach worked quite well. We used a value of $G = 500$ for the initial evaluation and $G = 10,000$ for the subsequent re-evaluation (if it was performed). Since most individuals were unable to beat the best individual seen thus far, they were not re-evaluated.

Selection and Termination. We use standard fitness-proportional selection [8] with elitism (i.e., the population contains a copy of the best individual that has ever been seen). For a termination criterion we ran the EA for a user-defined number of generations (2500).

5 Experimental Evaluation

We performed two experiments to judge the efficacy of our method. We were interested in answering two questions. First, how many states should be accessible initially? Second, does the adaptive-state EA find the optimal range of accessible states? To address the first question we ran an experiment where each FSM individual is initialized with $S = 10$ states. The experiment consisted of a comparison between the adaptive-state EA in three configurations: one, five, and ten initially accessible states. The only mechanism for adapting the number of accessible states is via the independent mutation operation mentioned above, which flips the accessibility tags. Although we have no "penalty" function per se (that would penalize the FSMs for having more accessible states), the mutation operator provides a slight bias towards having $S/2$ accessible states.

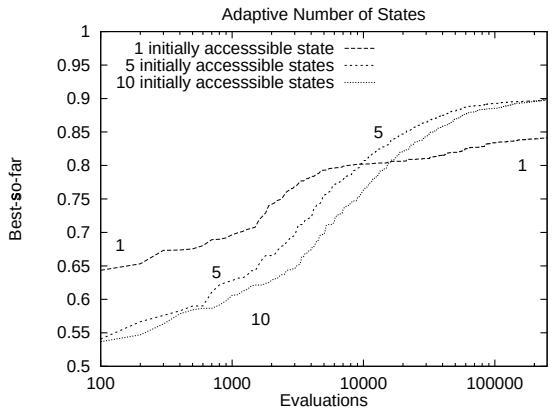

Fig. 1. "Best-so-far" curves for the adaptive-state EA with one, five and ten initially accessible states.

Figure 1 shows the best-so-far curves (the fitness of the best individual seen thus far) for the adaptive-state EA initialized with one, five and ten accessible states. The log plot emphasizes early behavior. Results are averaged over ten independent runs per configuration. On average the adaptive-state EA changed the number of accessible states in the FSMs, from one to five, five to seven, and ten to nine, respectively. One can see that having fewer initially accessible states helps early performance.[3] This is intuitively reasonable, since the adaptive-state EA is initially searching a smaller space in this situation. However, having too few

[3] The difference between one and five initially accessible states is statistically significant ($p < 0.04$) everywhere except between 7,000 and 18,000 evaluations. The difference between five and ten initially accessible states is significant ($p < 0.04$) between 2,000 and 19,000 evaluations. The data may not be normally distributed – hence we used an exact Wilcoxon rank-sum test with paired data in this paper.

initially accessible states (e.g., one) hurts later performance. With five and ten initially accessible states performance was quite reasonable, with a final fitness of 0.898 and 0.899 respectively. These results indicate that if the best number of states for solving a problem is not known a priori it may be best to err on the side of having too many, rather than too few.

The second question (whether the adaptive-state EA finds the optimal range of accessible states) is also important, although we have not seen it addressed in the literature. To address it we ran a control (ablation) experiment, where we turned off the adaptation of the number of states. Instead, the EA was run with a fixed number of states S. There were ten configurations (S ranged from one to ten) and ten independent runs per configuration. The results are shown in Table 1. Two points are clear. The first is that for best performance this problem requires FSMs with at least three states (i.e., state information is useful). The second is that performance is fairly comparable in the range of three to ten states.[4] This agrees with the previous experiment (with the adaptive number of states), which always ended in the range of five to nine accessible states.

	Fixed Number of States									
	1	2	3	4	5	6	7	8	9	10
Fitness	0.806	0.867	0.893	0.888	0.901	0.903	0.899	0.905	0.896	0.883

Table 1. The final fitness of the best individuals for each configuration, averaged over ten runs per configuration. The optimum number of states is between three and ten.

In summary, the adaptive-state EA effectively converges to the optimal range of states. Furthermore, its performance is competitive with the best fixed-state results (when started with five or ten initially accessible states). The only cause for concern is that although the adaptive-state EA that started with one initially accessible state ended with roughly five accessible states, the end performance was much poorer (0.841) than the fixed-state results (0.901). This suggests that although states are being made accessible the FSM is not taking full advantage of them. We investigate this possibility further in Section 7.

6 External Behavior of the Evolved FSMs

In order to understand and improve our results, we watched the agents play the simulation. The most noticeable feature was unproductive cycling behavior by the defender. In other words, the defender repeatedly visits a small set of squares on the board, while the adversary continues capturing new squares. Unfortunately, evolution alone can not solve this problem because cycles are inherent to FSMs. Therefore we augmented the FSMs with an auxiliary memory

[4] The increase in performance from one to two states and two to three states is significant ($p < 0.003$ and $p < 0.04$, resp.) whereas the other differences are not significant.

and an algorithm to use this memory to detect and eliminate cycles (up to a user-defined maximum length). To detect cycles, we use *behavior checking* (earlier results with *model checking* are described in [13]). Behavior checking examines the dynamic run-time behavior of the agent. Run-time checking of system behavior is a very new topic in the verification community, but some of the results already appear promising (e.g., [6]). Here, we present the first algorithm of which we are aware that does a run-time check for an FSM agent's cyclic behavior.

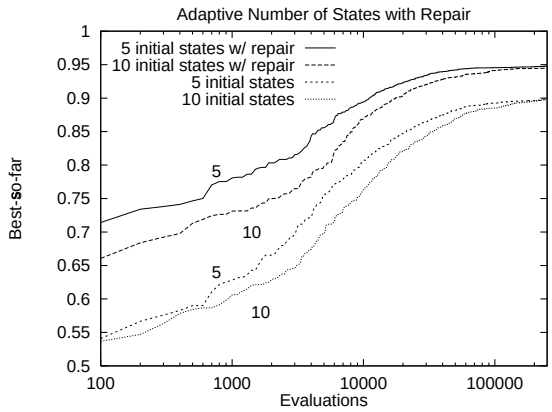

Fig. 2. A comparison of performance with and without cycle checking and repair.

Our behavior checking algorithm is executed while the agents play a game. For a sliding window of t time steps, the defender saves its current state and location on the board (the defender now consists of an FSM and auxiliary memory). The defender uses this auxiliary memory to make a cycle check before every move. If all four immediate neighbors are occupied, then the defender checks whether its current state and location are equal to any other in its window. If yes, a cycle has been identified and a random alternative action is taken to the one recommended by the FSM.[5] We have found that a window size of $2 \times N = 20$ time steps, which identifies cycles up to length 20, works well.

To test the hypothesis that behavior checking and cycle repair will improve performance, we reran the adaptive-state EA experiment with five and ten initially accessible states, but added in cycle detection and repair. This hypothesis is confirmed, as shown in Figure 2.[6] Our best performing defender FSM with repair wins 96% of the games!

[5] Of course, this alternative action could also create a cycle, but the behavior checking algorithm will immediately detect that cycle, after that move.

[6] The improvement using detection and repair is statistically significant, $p < 0.01$.

7 Internal Behavior of the Evolved FSMs

Although we achieved excellent performance with the addition of cycle detection and repair, we were still concerned that the adaptive-state EA might not be making good use of newly accessible states. To investigate this further we performed a dynamic internal analysis of the FSMs as they were executed by the defender. While the FSM was executing we counted the number of times that each of the $n \leq 10$ accessible states was actually the next state of a transition. The results for the fixed-state experiment were reassuring – the FSM tended to make reasonably uniform use of all n states. However, this was not true for the adaptive-state experiment. As stated earlier, on average the adaptive-state EA changed the number of accessible states in the FSMs from one to five, five to seven, and ten to nine, respectively. However, the internal analysis shows that only two, four, and six states (respectively) are actually being used to any appreciable degree. This explains the poor performance of the configuration where only one state is initially accessible. Although five states became accessible, only two were actually being used. Clearly the adaptive-state EA is having difficulty making full use of newly accessible states that have never been seen before.

This raises the serious concern that the addition and deletion of states are such disruptive operations that they cause noticeable problems for the evolution of FSMs. Currently we are investigating the application of "gentler" operators that could perform the same role. Simply deleting states (or turning them off) is too disruptive, due to the repair that must be performed afterwards. However, merging two similar states could remove a state in a fashion less deleterious to evolution. This process would be analogous to generalization. Similarly, as opposed to adding states (or turning them on), an alternative operator would clone an existing row of the tabular representation. Accessing this new state would not be deleterious and evolution could proceed to modify it slowly. This provides a process of specialization. We are currently exploring these options.

8 Summary and Future Work

To summarize, this paper has empirically explored issues related to evolving FSMs in the context of the Competition for Resources problem. Our experiments yielded some interesting and useful results. For example, given enough initially accessible states, it was encouraging to find that the adaptive-state EA was able to successfully converge to the optimal range of number of states and was able to provide good performance. However, problems arise when starting with too few initially accessible states, and an analysis indicates that (for the Competition for Resources problem at least) the adaptive-state EA is having some difficulty making good use of newly accessible states. We also found that the ubiquitous presence of cycles hampered the defender's performance significantly. This latter difficulty was greatly diminished by augmenting the FSMs with memory and an algorithm for cycle detection and repair.

Our main focus for the future is to improve the evolution of the FSMs, to make the Competition for Resources game more realistic, and to continue

our empirical investigations in the context of newer versions of the game. For example, in the current game resources are all treated equally. In the spirit of game theory, we would like to consider resources having different numeric values, and perhaps have the value of a resource differ for each of the agents. Another possibility is to allow one agent to (with some small probability) "steal" a resource owned by the other agent. Another possibility is to include multiple agents and co-evolution. What is most interesting about this game is how easily it can be changed to represent a wide variety of problems. For example, with minor modifications we have extended the game to represent the epidemiology of virus versus anti-virus spread. In the virus version of the game, each square represents an agent with the virus, anti-virus, or neither. At each time step, an agent having the virus or anti-virus can spread it to one of its neighbors. What one sees on the board when watching this version of the game looks like a "spreading activation". Further pursuit of the virus version both in simulation and in a corresponding mathematical model are currently in progress.

References

1. Carmel, D. and Markovitch, S. (1996) Learning models of intelligent agents. Proceedings of the Thirteenth National Conference on Artificial Intelligence.
2. De Jong, K., Spears, W. and Gordon, D. (1993) Using genetic algorithms for concept learning. Machine Learning Journal, 13 2/3.
3. Denning, D. (1999) Information Warfare and Security. Addison-Wesley, NY.
4. Fogel, L. (1999) Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming. Wiley Series on Intelligent Systems.
5. Fogel, L., Owens, A., and Walsh, M. (1966) Artificial Intelligence Through Simulated Evolution. John Wiley and Sons, Inc., New York.
6. Gordon, D., Spears, W., Sokolsky, O., and Lee, I. (1999) Distributed spatial control, global monitoring and steering of mobile physical agents. Proceedings of the IEEE International Conference on Information, Intelligence, and Systems.
7. Grefenstette, J. and Fitzpatrick, J. (1985) Genetic search with approximate function evaluations. Proceedings of the Int'l Conference on Genetic Algorithms.
8. Holland, J. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press.
9. Hopcroft, J. and Ullman, J. (1979) Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Menlo Park.
10. Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C. and Wang, A. (1991) Evolution as a theme in artificial life: the Genesys/Tracker system. Proceedings of Artificial Life II.
11. Levy, S. and Stone, B. (2000) Hunting the hackers. Newsweek, February 21.
12. Spears, W. and De Jong, K. (1991) On the virtues of parameterized uniform crossover. Proceedings of the International Conference on Genetic Algorithms.
13. Spears. W. and Gordon, D. (2000) Evolution of strategies for resource protection problems. Submitted to A. Ghosh and S. Tsutsui, editors, Theory and Application of Evolutionary Computation: Recent Trends. Springer Verlag.