Learning Mental Models

Astro Teller
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
astro@cs.cmu.edu

ABSTRACT

The process of learning is not always as simple as mapping inputs to the best outputs. Often internal state is needed to distinguish between observably identical states of the world. Genetic programming has concentrated on solving problems in the functional/reactive arena, in part because of the absence of a natural way to incorporate memory into the paradigm. This paper presents a simple addition to the genetic programming paradigm that seamlessly incorporates the evolution of the effective gathering, storage, and retrieval of arbitrarily complicated state information. Experimental results show that the effective production and use of complex state structures can be evolved and that agents evolving the use of memory quickly and permanently displace purely reactive and non-deterministic functions. These results may not only aid future research into the causes and constituents of mental models but may expand the types of problems that can be practically tackled by genetic programming.

INTRODUCTION

Imagine a small boy (or any small agent for that matter) riding a bicycle around his neighborhood. We might say that this agent has "learned" to ride the bicycle. This learning process is highly reactive. The ability to lean in different directions in order to remain upright requires no state information. We might also say that the agent has "learned" its way around the neighborhood. This learning is not only the developed ability to navigate but the construction of a mental map of the area.

Machine learning as a discipline often emphasizes problems of classification and the discovery of functions. In general, an agent can neither act nor learn in an environment without some model of the environment. This model must come as a combination of those model features given to the agent and those it builds and maintains on its own. Learning, given an environmental model, is an important problem. Creating a model, however, is also a fundamental part of what it means to learn. The machine learning technique called genetic programming has been applied in the past to problems like the ability to balance on a bicycle but little work in evolutionary strategies has been done on creating and using a mental model of the environment.

Problems requiring memory or state have been attempted in both genetic algorithms and genetic programming. Both areas have used state implicitly and explicitly. In no case, however, has it been a natural, integrated part of the process. In the field of genetic algorithms encoding have been done for both finite state automata and recurrent neural networks. (Jefferson et al. 1992) In both cases, binary encoding of lengths over 400 lead to 32 states. In an evolution of solutions to the Prisoner's Dilemma using state, the binary encoding used 2^m bits for an individual who could remember back m binary moves. (Lindgren 1991) In genetic programming, one of the techniques has been to use *Progn* as a way to string together side-effecting terminals. (Koza 1992) This is a more implicit use of state. It is not clear whether this use of state in genetic programming would be generally applicable in memory intensive problems and it does not lend itself to any kind of introspection (i.e. decision making based on a function's current state).

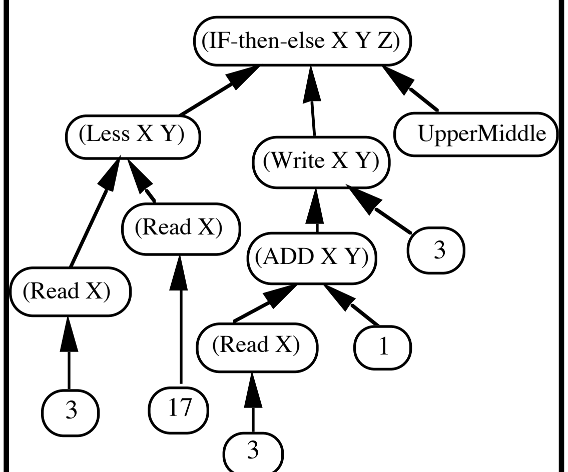
The purpose of this paper is twofold. Foremost, it represents an attempt to evolve agents that use state effectively and even build mental models of an environment given only temporally and spatially immediate sensory input. These mental model builders use an addition to the genetic programming paradigm called indexed memory. This addition is evaluated with respect to this research and as a potentially general tool for using memory in genetic programming.

Indexed memory was not designed specifically to foster the emergence of mental models, but to be a general method for the use of memory in the field of genetic programming. This paper, through results and discussions, will suggest that this new scheme for incorporating the use of memory into genetic programming will be generally applicable to a wide range of problems that have already been investigated and will open the door to new problem areas that are memory-critical. In addition, this attempt to produce agents that build up and effectively use state may be a small step into using evolution to study what mental models are and how, or even if, we can detect them.

THE METHOD AND MODEL

In the general case of indexed memory, each population individual has not only a functional tree, but an array (Memory) of elements indexed from 0 to (M-1). Each of these M elements is an integer in the same range. Two non-terminals are added to any existing function set: Read and Write. Read takes one argument (Y) and returns Memory[Y]. Write takes two arguments: X and Y. It returns the old value in Memory[Y] and puts X into Memory[Y]. Both parameters can be any integer between 0 and (M-1) and both functions return an integer between 0 and (M-1). For the Tartarus world (described below) M was chosen to be 20. The agents have 20^{20} states available to them (approximately $1.05 * 10^{26}$). This is significantly larger than the few dozen states available to most genetic algorithm and genetic programming experiments in the past.

Main Branch or Subtree



The tree shown here could be the whole tree or a subtree of a main function. This picture highlights a use of indexed memory. In English this function branch would translate into:

"If Memory[3] < Memory[17] then re
Memory[3] and add 1 to it. Otherwise
the input value of the UpperMiddle se

The returned value from Memory[3] is its value before the increment. Memory[3] is not always incremented because the tree is not fully evaluated. For example, the test branch of the IF-THEN-ELSE is evaluated and then, based on the result, either the then clause or the else cause is evaluated, but not both.

There are only two kinds of terminals: constants (between 0 and (M-1)) and inputs. Indexed memory constrains all functions to return integers between 0 and (M-1) so that any computed value is a legal memory index. The non-terminals selected for this problem were:

(OR X Y) --> This returns the MAX(X,Y)
(NOT X) --> If X=0 Then 1 Else 0
(LESS X Y) --> If X<Y Then 1 Else 0
(ADD X Y) --> Returns X+Y (ceiling of M-1)
(IF-T-E X Y Z) --> If X>0 Then Y Else Z
(SUB X Y) --> Returns X-Y (floor of 0)
(EQ X Y) --> if X=Y Then 1 Else 0
(Read Y) --> Returns Memory[Y]
(Write X Y) --> Returns old Memory[Y]
(side-effect: Memory[Y] <==X)

In addition to the main genetically programmed functional tree, the agent is allowed an automatically defined function (ADF) tree with two arguments. In that tree the terminals and non-terminals already stated are legal. The ADF has two additional terminals: Parameter1 and Parameter2. The main function of the agent has the additional non-terminal (ADF X Y) which sets Parameter1 to the evaluation of the subtree X and Parameter2 to the evaluation of subtree Y and then evaluates ADF. Like the other functions, ADF returns an integer in the range of 0 to (M-1).

THE ENVIRONMENT

Tartarus is an NxN grid. Approximately $(N-2)^2/3$ boxes are randomly distributed on the inner (N-2) x (N-2) grid. The agent can be in any square that does not have a box and may face North, South, East, or West. N was chosen to be 6 in order to maintain computational tractability and to maximize certain criteria including the number of different possible initial configurations and low box density. This means that there are 6 randomly placed boxes on the inner 16 squares. At time step 0 of each fitness test, each agent finds itself on a random clear location facing in a random direction. (e.g. Figure 1)

The agent is given 8 input sensor values for the eight nearest neighbors to the grid position it inhabits (Figure 2). These 8 inputs can take on one of three values: Clear==0, Box==1, or Wall==2. The Wall value (2) indicates that the position is outside the boundaries of the world. These eight input values can be used in the genetically programmed functions as eight terminals called {UpperLeft, UpperMiddle, UpperRight, MiddleLeft, MiddleRight, LowerLeft, LowerMiddle, LowerRight}. These inputs are relative to the agent's heading.

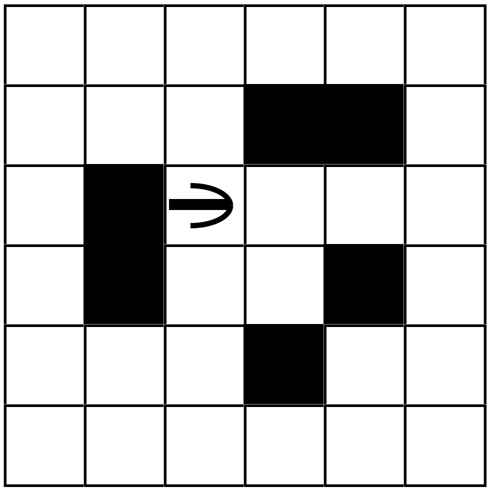


Figure 1

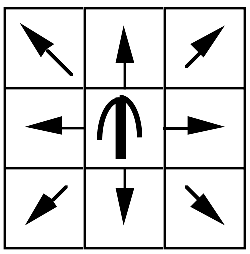


Figure 2

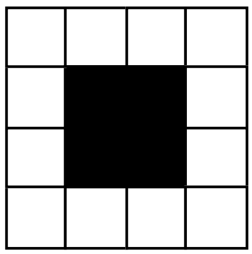


Figure 3

The agent's goal is to push all of the boxes out of the center onto the outer perimeter of grid positions. At the end of a fitness test the agent is rewarded 2 points for every corner that contains a box and 1 point for every non-corner edge position that contains a box. The agent has only three possible actions: MoveForward, TurnLeft, and TurnRight.. Moving forward is allowed under two conditions: when the grid position in front is clear and inside the world boundary or when the position in front has a box but the space after that (in the same direction) is clear and inside the world boundary. In the latter case the agent moves to the next square and the box is pushed one square forward. If the agent attempts to move forward into a wall or into a box which may not be pushed the action fails and the environment remains unchanged. Turning left, right, and moving forward each take 1 time step. One complete tour of the board would take $N^2+2N-3 = 45$ and each agent is allowed only 80 time steps. The main function for each agent does not return {MoveForward, TurnLeft, TurnRight }. It returns integers in the range 0 to (M-1). A filter is needed to map these numbers into one of the 3 possible actions. For M=20 the mapping was { Move (ReturnValue < 7), Left (6 < ReturnValue < 14), Right (14 < ReturnValue) }.

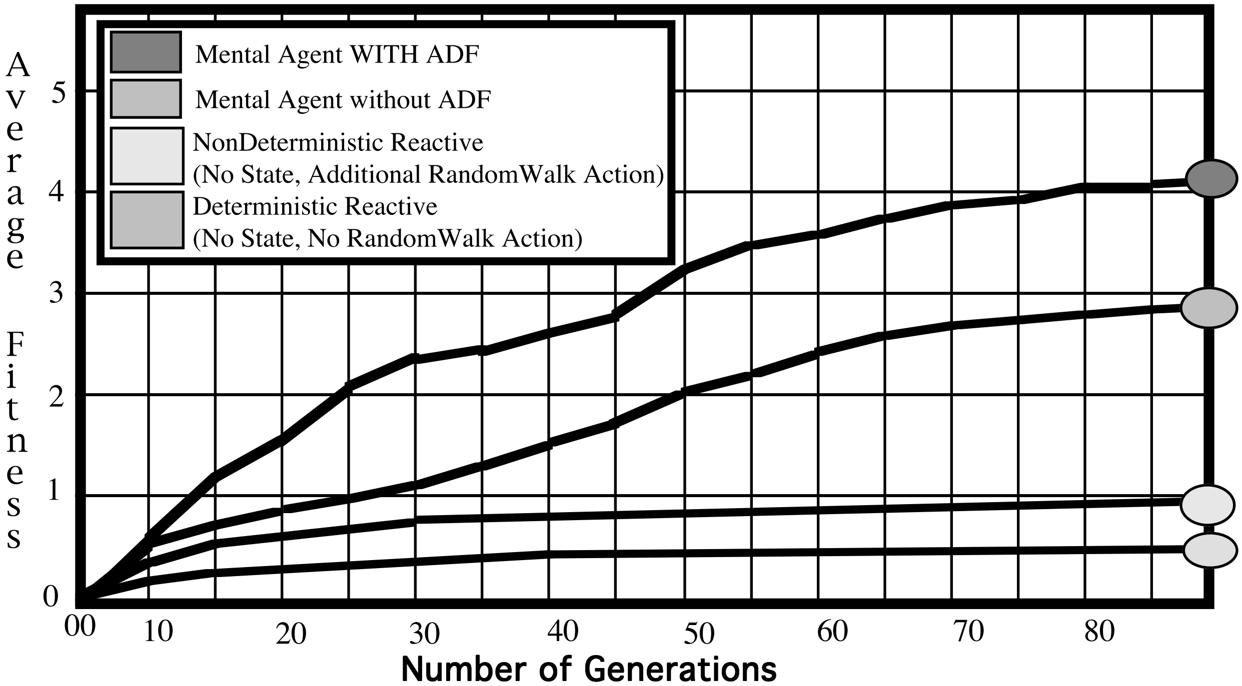
Tartarus was designed to promote the intelligent use of state. The intention was to make a task with sufficient complexity in an environment with sufficiently random initial conditions so that agents who use memory would have a sizeable advantage over agents who ignore their memory or who do not have memory. It cannot be overstressed how hard it is to achieve high fitness in the Tartarus environment without memory. Considerable effort has been put into writing a function that can reliably get more than 1 point every 80 time steps without using indexed memory. No human-generated or machine-generated program has yet materialized. The non-locality of relevant time and space world features is a fundamental reason why Tartarus is a good environment in which to promote the use of state.

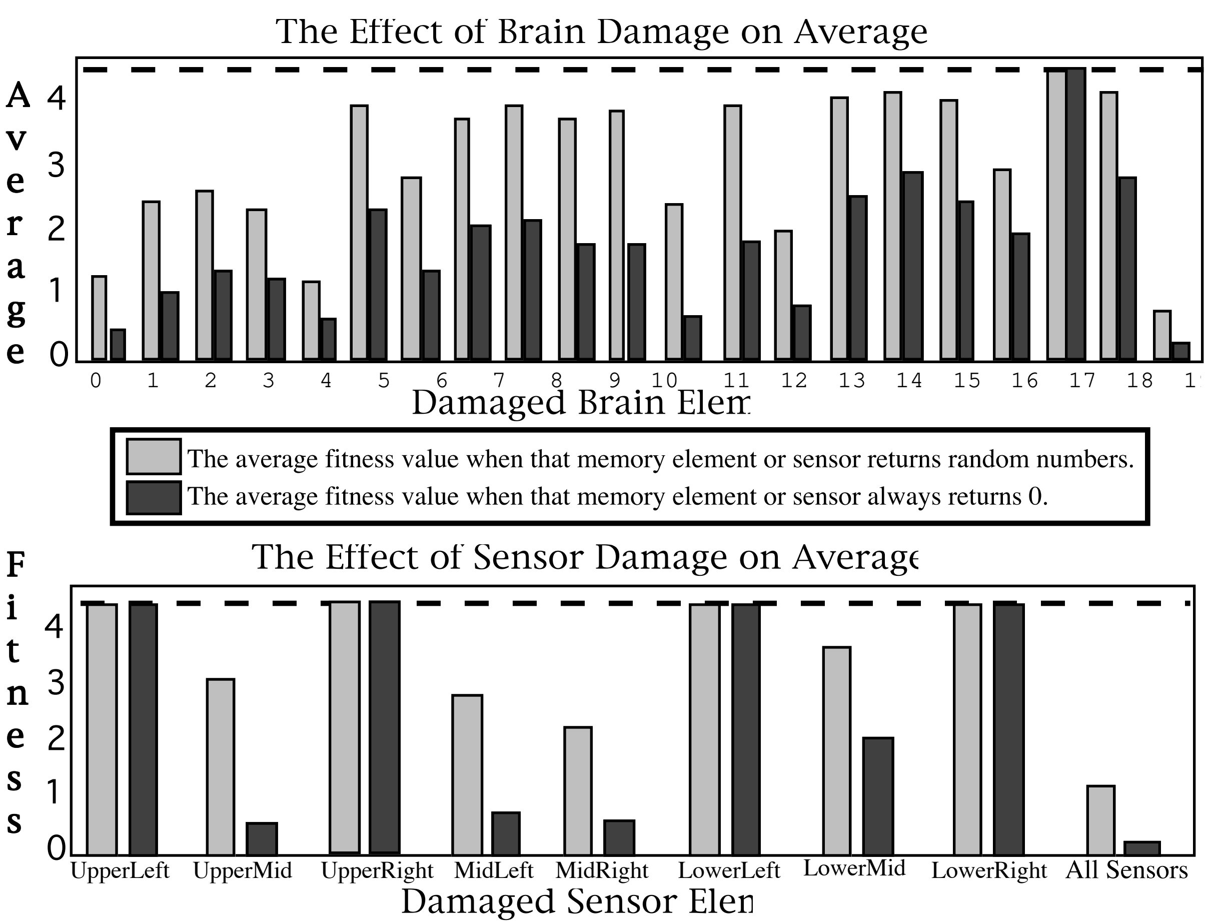
THE EXPERIMENTS

So far this paper has concentrated on a new method for incorporating memory into the genetic programming paradigm. A type of population individual was described and the following results focus on the success of this kind of agent. In order to fairly asses the merit of this type of agent, three benchmark agent types are included for comparison in the runs.

The first graph on the next page shows the average best of generation for each of the four types. This average was taken across 10 runs, each of which was allowed to go to generation 80.

The most important observation to take away from the first graph is that the agent types with indexed memory do much better than the agent types with no access to state. The base benchmark agent (Deterministic Reactive), as expected, is the least successful. After 80 generations, its best of generation is getting an average of about half a point per fitness test. The random agent (Non-deterministic Reactive) does almost twice as well, but that still amounts to very little. After 80 generations it does not consistently get even one point per fitness case. While we can see that the ADF is very useful in this domain, it clearly is not the most dominant issue. The difference between the mental agent with ADF and the deterministic reactive is a factor of 8.5. The difference between the mental agent type without ADF and the deterministic reactive agent type is about a factor of 6. But the difference between the mental agent type with ADF and the mental agent type without ADF is not even a factor of 2. The conclusion is that state information is critical to getting high fitness in this domain and that indexed memory is an effective evolutionary tool in achieving this use of state information.


Mental Agent WITH ADF
Mental Agent without ADF
NonDeterministic Reactive
(No State, Additional RandomWalk Action)
Deterministic Reactive
(No State, No RandomWalk Action)
Average Fitness
0
1
2
3
4
5
00
10
20
30
40
50
60
70
80
Number of Generations


The Effect of Brain Damage on Average
Average Fitness
0
1
2
3
4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Damaged Brain Elem
The average fitness value when that memory element or sensor returns random numbers.
The average fitness value when that memory element or sensor always returns 0.
The Effect of Sensor Damage on Average
UpperLeft
UpperMid
UpperRight
MidLeft
MidRight
LowerLeft
LowerMid
LowerRight
All Sensors
Damaged Sensor Elen

Determining if agents are using their memory in ways that correspond to particular features of the world is a difficult, open-ended process. In order to determine some of these more subjective qualities, it was necessary to perform experiments on a specific agent. The second and third graphs on the previous page are data taken about a typical mental agent type with ADF that was the best of a generation 80. Its average fitness was 4.4 points per fitness test. Here is the s-expression that represents that individual:

Automatically Defined Function

(WRITE (IF-THEN-ELSE (ADD (WRITE 0 Param1) (OR Param1 (READ (WRITE (READ 0) (READ (ADD Param2 (ADD Param1 (READ 0))))))))) (ADD Param2 (ADD (ADD (READ (READ (READ 0))) (ADD Param1 Param2)) (ADD Param1 (READ MidRight))))) (WRITE (WRITE (WRITE 0 (WRITE (WRITE (OR (READ (WRITE Param1 Param2)) 0) Param1) (WRITE Param1 Param2)))) Param1) (WRITE (WRITE (READ 0) (READ (WRITE Param1 Param2)))) (WRITE Param1 0))))) (READ (READ (READ (READ (EQ Param1 Param2))))))

Main Function

(ADD MidLeft (WRITE (SUB 3 (NOT (OR UpperMid (EQ 2 (IF-THEN-ELSE (SUB (SUB (READ 17) UpperMid) 8) (*A-D-F* (EQ UpperMid (IF-THEN-ELSE (READ 17) UpperMid 10))) (WRITE 10 UpperMid)) (LESS (WRITE UpperMid UpperMid) (*A-D-F* LowerMid (WRITE UpperMid UpperMid))))))))(WRITE 8 UpperMid)))

The second graph on the previous page is an account, over many fitness tests, of how this particular agent fared with a form of brain damage. In each set of runs, some memory index i was selected and that memory index always returned a random number between 0 and 19 no matter what the actual value was. This was done for 100 random fitness tests, and an average fitness was found for the individual with that brain element damaged. This was done for each index i between 0 and 19. It is clear that this individual relies heavily on the values stored in certain memory positions. The same experiment was performed with the damaged memory index returning 0 independent of the true memory value. This caused greater reduction in the fitness of the individual. When the all of the memory indices were simultaneously subjected to the random or constant brain damage, the individual's fitness dropped to around 10% its normal fitness.

In the third graph on the previous page we see the results of the complement to the brain damage study. This study subjects the same individual to sensory deprivation. Blinding each sensor input individually, 100 fitness tests were done and the histogram shows the resulting fitness averages. Each blinded sensor returned a random sensor value in the set {0,1,2}. The same tests were rerun and each blind sensor returned 0 regardless of the correct input. Here, as with the brain damage study, the sensor constant misinformation caused a greater loss of fitness than did the sensor random misinformation. When all the sensors were blinded simultaneously the individual's fitness dropped to 20% and 5% of the normal fitness (for the random and constant blinding respectively).

DETAILS OF IMPLEMENTATION

Population	8
% Crossed Over	
% Crossed Over in Main Function	
% Crossed Over in ADF	
% Subjected to Point Mutation	
Number of Fitness Cases used to determine fitness	
Maximum Depth allowed in Main Function	
ADF MaxDepth	
Machine used	DEC5000/1
Average hours to finish generation 80	1

DISCUSSIONS

One of the advantages of the indexed memory strategy is its large number of possible states. Consider that for fewer than two dozen bytes of storage the agent gets over 10^{26} possible states (M=20) with constant access to 20 cross sections of the state space. Redundancy brings with it a certain flexibility. This flexibility is the increased number of possible representations of a set of states. Since in using memory, the evolutionary process is trying to find some workable representation of the useful environmental features, more possible representations is better. Since we pay almost nothing for all the extra state in indexed memory, we effectively get the flexibility for free.

The efficacy of indexed memory is affected by the choice of M=20. Several other choices for M were investigated. For values of M less than 8 or 9 the performance noticeably decreased. For values of M larger than 40 or 50 there was also a gradual decrease in the rate of increase of average fitness over generations. For low M the performance loss is probably attributable to insufficient memory space. For large M the opposite problem occurs: there are too many memory elements. As a result, in early generations there is a lower likelihood of reading from the same memory element that useful information was written to.

Indexed memory should work for a wide variety of problems requiring state or mental models. For example, the Prisoner's Dilemma problem has already been mentioned as a problem for which memory is useful. Indexed memory could easily accommodate keeping the last k moves as long as k < M and could probably find encoding that would work for k>M. As an example the function could always start by moving Memory[2] into Memory[1], Memory[3] into Memory[2], ... until Memory[k] was moved. Then it could put its most recent input (the past move) into Memory[k]. Another strategy could be to use a circular queue and leave out Memory[0] and Memory[1] as pointers to the head and tail of the queue. These two examples show that indexed memory is a sufficiently powerful tool for the problem. Undoubtedly the evolutionary process would come up with an equally valid technique that would not map into any familiar algorithm.

FUTURE WORK

The results described in this paper lead toward at least two different lines of the research. The first is the use of techniques, like the one presented here to create or evolve mental models and to design experiments to study these models. An interesting step would be to make some distinction between simple use of state and a mental model based on phenomena observed in the memory units. The second line is to continue to improve upon indexed memory as a paradigm in genetic programming or to expand this work in some natural way into genetic algorithms. The introduction of natural uses of memory in these environments may eventually lead to a broader range of problems that can be learned using evolutionary strategies.

To substantiate the claim that indexed memory is truly a general solution to the problem of incorporating memory into genetic programming, it must be tried on a variety of other memory-critical problems. Work has already been started using indexed memory and the same non-terminal set discussed here, to find solutions to the problem of learning a maze, the Prisoner's Dilemma, and a modified version of Simon-Says.

CONCLUSIONS

An important goal of this research effort was to evolve agents that use state effectively and build mental models of their environment. As was shown using the brain damage and sensor damage graphs, the loss of particular memory elements or sensor elements results in significantly lower fitness. This means that memory and sensory inputs play a pivotal role in that individual's success. It was argued that a dependence on inputs that correspond to the real feature values from the environment, and a dependence on the consistent ability to retrieve from memory the same value that was placed there, implies some complex use of memory.

These results were produced using the new technique of indexed memory. Explanations were given for the choice of the particular index used in these experiments and relative costs were given, in this domain, for increasing or decreasing the maximum index value. Work in progress includes the use of indexed memory to solve several familiar memory-based problems and suggestions were given about how indexed memory might be used by evolving systems to solve problems in other domains.

It has been argued that this research saw the emergence of mental models in agents that evolved from functional agents with access to indexed memory. These behaviors could not have come about without some type of simple, malleable, state information that could be used in a variety of ways. As a representative of this type of state information system and as an addition to the genetic programming paradigm, indexed memory stands in a position to advance work done on the genesis, types, and structures of mental models and to open up new types of problems to the field of genetic programming.

ACKNOWLEDGEMENTS

I benefited greatly from discussions I had on this subject with John Koza, Nils Nilsson, and Ronny Kohavi. The original motivation for this research came from a problem John Koza and Nils Nilsson brought to my attention which they call "the 7-layer Cake" problem. It was in quest of progress on this domain, the hierarchy of memory usage, that this paper emerged

BIBLIOGRAPHY

Collins, Robert, Jefferson, David. 1991. AntFarm: Toward Simulated Evolution. In Langton, Christopher, et al. (editors), Artificial Life II. Addison-Wesley.

Jefferson, David, Collins, Robert, Cooper, Claus, Dyer, Michael, Flowers, Margot, Korf, Richard, Taylor, Charles, and Wang, Alan. 1991. Evolution as a theme in artificial life: The genesys/track system. In Langton, Christopher, et al. (editors), Artificial Life II. Addison-Wesley.

Koza, John R. 1992. Genetic Programming: On the Programming of computers by means natural selection. The MIT Press.

Lindren, Kristian. 1991. Evolutionary Phenomena in Simple Dynamics. In Langton, Christopher, et al. (editors), Artificial Life II. Addison-Wesley.