Using Experience-Based Learning in Game Playing[1]

Kenneth A. De Jong dejong@nrl-aic.arpa
Alan C. Schultz schultz@nrl-aic.arpa

Navy Center For Applied Research In Artificial Intelligence, Naval Research Laboratory, Washington, DC 20375-5000 U.S.A. and
Department of Computer Science, George Mason University, Fairfax, VA 22030 U.S.A.

ABSTRACT

This paper describes some initial experimental results in the use and limits of applying *experience-based* learning techniques to the area of game playing. A system (GINA) has been developed that uses experience-based learning to improve its own base performance level for the game of Othello. A variety of traditional (non-learning) Othello game playing programs have been collected to provide a rich source of ''experiences'' for GINA. An initial set of experiments has been designed and run in an attempt to understand better the strengths and weaknesses of experience-based learning from the point of view of skill refinement (taking less time to perform at a given level of play) and from the point of view of skill enhancement (higher quality play). Preliminary results suggest that in this context experienced-based learning can be a highly effective means for improving both the speed and quality of play without having to accumulate experience bases of intolerable size.

1. INTRODUCTION

We use the term *experience-based learning* to characterize the kind of learning that takes place as a consequence of attempting to perform a complex task, evaluating the results of that attempt, and ''remembering'' certain aspects of that experience in order to ''improve'' subsequent task performance. Humans clearly exhibit this kind of learning in tasks as diverse as medical diagnosis, fault isolation in man-made systems, theorem proving, writing programs, and playing games. The authors' interest in experience-based learning comes from *their own* experiences in designing and implementing (non-learning) second-generation expert systems for fault isolation of complex, man-made systems.[1] It became quite clear that the long-term usefulness of such expert systems *in the field* would hinge on the ability to include an experience-based learning mechanism, rather than depend on an omniscient knowledge base or continued manual updating of the knowledge base.

As a consequence, we have an ongoing research effort involving the development of a general architecture for experience-based learning problems of this sort, and evaluating the strength and robustness of this approach in a variety of problem domains. This paper provides a brief description of the current architecture and presents some initial empirical results from the application of this approach to the area of game playing.

2. SOME GENERAL DESIGN ISSUES

We begin by attempting to characterize the important features and assumptions of the learning problem under study. First, we are interested in improving problem solving skills which consist of making a series of complex decisions in an attempt to attain a goal. Second, we assume that in

[1] From *Proc. of the Fifth International Machine Learning Conference*, June 12-14, 1988, Ann Arbor, Michigan, pp. 284-290.

general there will not be feedback available concerning the correctness or quality of individual decisions. Rather, the only feedback will be in terms of the final outcome, leaving to the learner the burden of any *apportionment of credit* to particular elements of the decision chain. Third, in order to achieve any hope of a general learning mechanism usable with quite different problem solvers, descriptions of experiences are restricted to contain only externally observable behavior (namely, situations encountered, decisions that were made and final outcome). No information about the internal behavior of the problem solving apparatus is available. Finally, we assume that the learner will have only a certain amount of indirect control over the set of experiences available. That is to say, although the learner might select a particular opponent to play or a particular system to fault isolate, there is no way to control the actual outcome of that experience.

Given this general characterization of the learning problem, we now briefly discuss the applicability of some existing familiar learning methods. The kind of real-world problems motivating this work do not currently provide the strong domain theory required to apply explanation-based and other deductive learning strategies.[2] The fact that the problem under study is one of performance improvement of a lengthy, sequential, decision making process prevents any direct application of the concept formation, classification, and similarity-based approaches.[3,4] Using experience to refine an evaluation function,[5] although appropriate for game playing, does not seem particularly useful for other tasks such as troubleshooting. The methods most closely matched to the present problem would seem to be the chunking mechanism in SOAR[6] and the LEX system.[7] However, both approaches assume access to the internal workings of a particular style of problem solver.

As a result of these observations, we have chosen to explore an approach which evolves over time an experience base which is available to the problem solver *as an augmentation* to its static, built-in capabilities. The basic idea is that a problem solver, when faced with making a decision, first consults with its experience base to see if any useful advice is available. If not, it falls back on its pre-programmed method for making a decision. When the task is completed (aborted, etc.), the learning component uses the description of this experience (what situations were encountered, what decisions were made, and the final outcome) to update the experience base.

The experience base is designed to represent knowledge in a form easily modified by the learner and quickly accessed by the problem solver. As the experience base grows, the ‘‘hit rate’’ of getting useful advice from the experience base increases which, in turn, increases both the speed with which decisions are made as well as the quality of those decisions. Notice that, although there is some superficial resemblance to rote memorization and cacheing techniques like memo functions,[8,9] the structure of the experience base and the character of the learning mechanism must be considerably more complex since there is no feedback concerning the quality of individual decisions.

We feel that, if achievable, such an approach has two attractive features. First, one can easily augment existing non-learning systems without having to understand and make significant changes to their internals. Second, although the specific internal representation of the experience base may be problem specific, the methodology is not and can be easily moved to other applications.

3. THE STRUCTURE AND CONTENT OF AN EXPERIENCE BASE

Although the previous section described the general approach, two issues require further specification: the format of an experience base and the learning procedure for integrating new experiences. In this section we discuss these issues in a problem independent framework.

An experience base is a directed graph of nodes (frames), each of which is intended to summarize what has been learned so far about a particular situation. New nodes are created and existing nodes are modified each time the learning component is provided with the description of an

experience (in the form of a sequential list of situations encountered, actions taken, and the final outcome). Since actions taken result in a (possibly) new situation, they correspond to directed arcs between nodes. In general, such networks can not be assumed to be tree structures, since nodes can have more than one parent and cycles can occur.

Because the experience base is intended to be interrogated by the problem solver for useful advice, a hash table is also maintained to provide quick access to nodes associated with a particular situation.

An experience base of the form described so far is capable of remembering responses to situations, but is not able to provide any information concerning the relative merits of responses, since feedback of this sort is only provided for the final outcome. The apportionment of credit to individual situation-actions is a thorny issue which has been around for a long time.[10][11] This is complicated in our case by the fact that a particular situation-action pair can appear on many experience chains. The general approach we have taken can be roughly described as a form of back propagation of the final outcome to the nodes involved using some ideas from game playing (see the section on GINA for more details). At this point in the research it is not clear to us whether a general mechanism of this sort will be sufficient for all applications, or whether some domain-specific method will be required.

Two other issues are also unresolved at this point: the need (if any) for a ‘‘forgetting’’ mechanism, and the need (if any) for a generalization mechanism. We plan to report on these issues as we expand our empirical studies.

4. GINA: A CASE STUDY USING OTHELLO

Although motivated by the need for experience-based learning in fault-isolation expert systems, we decided to initially explore our ideas in a sufficiently complex, but more controlled area of problem solving. The task selected was that of playing the game of Othello. The remainder of this paper describes our initial results in using the approach described in the previous sections to implement GINA, an Othello game playing program which *continuously* learns from its own experience each time it plays.

4.1. The Architecture of the System

In order to apply our architecture to game playing in general and Othello in particular, there were three key issues to be resolved: 1) how to represent ‘‘situations’’ in the experience base; 2) how to handle apportionment of credit; and 3) how to modify an existing game program to ‘‘consult’’ the experience base. Each of these issues is addressed in the next three sections.

4.1.1. The Experience Base

After considerable thought, we decided to represent situations as actual board configurations and the player to move. We felt that depending on feature vector encodings or other descriptive mechanisms would require too much pre-selection on our part and/or too much delving into the workings of a candidate base-level player. This means that GINA's experience base is essentially a partial game tree in which nodes represent actual board configurations encountered and arcs represent moves actually tried. (Note the difference between this approach and Samuel's earlier rote learning method[10] which learned a large number of board positions and the *correct* moves.)

The immediate reaction is one of concern about the ultimate size of an experience base. Our working hypothesis (confirmed by initial experiments) is that the number of *reasonable* board configurations *actually* encountered would be quite manageable. If not, mechanisms for forgetting

and generalization could be considered.

To be more precise, each node consists of the board configuration, various statistics, pointers to each child of that node, and a pointer to a linked list of pointers to that node's parents. The board configuration consists of a compact representation of the current board and the player to move. The statistics include frequency and min-max value. The frequency is the number of paths that exist in the structure below this node, and is available as an estimate of the reliability of the min-max value. Note that this is equal to the number of leaf nodes below this node in an equivalent tree structure. The min-max value is the backed-up min-max value from the terminal board positions of the actual games played. This value is available as an estimate of the desirability of the position from black's point of view.

The experience base is stored in compact form in a binary file and consists of a hash table and the directed graph. The hash table is copied into memory before use. The hashing function maps the board configuration to an entry in the hash table. The entry in the table gives the file address of the node associated with that configuration. The hash table is followed by the root node and then the rest of the nodes of the partial game ''tree''.

4.1.2. Adding Games to the Experience Base

During a game, the moves for each player are recorded in a file. When a game has been completed, the game file is incorporated into the experience base. Starting with the first move, the current board and player to move is used to hash into the hash table. If the node is not found, then a new node is created. In some cases, the node may exist, but not be on the current trajectory through the structure. In this case, a parent pointer of the existing node is added.

When a new node is created, the number of legal children is calculated, and slots (initialized to nil) are created for pointers to each child. A structure to hold pointers for each parent of the node is also created. Since the number of parents a node may have is not known, a linked list is used for these pointers.

After the appropriate nodes and links have been created for the entire game, it remains to calculate what we have *learned* from this experience. The only feedback available is the outcome of the game which can obviously be associated with the final board position leaf node. However, we also have additional indirect information about ancestors of this leaf node in the experience base (not just those involved in the game being integrated). Given the context, the most natural way to distribute this information was to back propagate from the leaf node the outcome of the game (expressed as black's stone advantage) to all ancestor nodes using a classic min-max algorithm. Hence, at any given time each node contains an estimate of its true min-max value based only on the games played, and a frequency count of how many distinct games contributed to that estimate as a measure of its reliability.

4.1.3. Using the Experience Base

As stated earlier, one of our design goals was to be able to add our experience-based learning component to an existing problem solver with minimal changes. In the case of GINA, we selected one of the better Othello playing programs we had access to and modified only the move selection code to first consult the experience base for move information. If no previous experiences exist, GINA uses its built-in strategy. The interesting case, of course, is what to do if the experience base contains information about the current situation. Note that there is no reason to believe that the min-max estimates in the experience base are any more reliable than the built-in move selector.

We decided that there was no one answer to this question, and depended on the kind of ‘‘personality’’ we wanted GINA to exhibit. The one we chose for these experiments was an aggressive, tournament-oriented personality out for maximum wins and exploiting the experience base whenever possible on the assumption it would quickly begin to reflect the opponents strengths and weaknesses. To be more precise, the following algorithm is used to decide on a move from the experience base (or in some cases, the fall back strategy). If there are one or more ‘‘good’’ moves from this board configuration in the experience base (ie. better than a tie game), then the one with the best value is used. If only bad moves exist, and all child nodes have been explored, then the child with the best value is used. However, if in addition to bad moves, unexplored moves exist (unexplored in the experience base), then the fall-back strategy is used to recommend a move. If the fall-back strategy's recommended move is one of the unexplored moves, the recommendation is used as a move. If, however, the recommendation is for one of the known moves, then the advice is ignored, and again the best existing move from the experience base is used.

This strategy has proven to be quite effective, but clearly does not exhibit other traits such as ‘‘curiosity’’. Other strategies are possible and these stategies will be explored in future research.

4.2. The Initial Experiments

The goal of these experiments was to discover to what extent the performance of an existing game-playing system could be improved by augmenting it with experience-based learning. There were two aspects of performance we were particularly interested in monitoring: improvements in the speed of play without loss of quality (skill refinement) and improvements in the quality of play (skill enhancement). In addition, since an experience base can be viewed as a space-time tradeoff, we were interested in how the experience base grows over time.

In order to gather reliable data, we collected a variety of different automated Othello players to be used as opponents. Most were typical computer Othello games using min-max search with alpha-beta pruning, differing primarily in their static evaluation functions. The first round of experiments consisted of tournaments of games between GINA and selected opponents, alternating who made the first move in each game. Each tournament was repeated a minimum of four times, and results shown average the tournaments together.

In each experiment, several statistics were gathered. GINA's quality of play (skill enhancement) was measured by the point advantage at the end of each game. To make sure there was room for improvement, GINA's built-in move evaluator was set to use 2-ply lookahead while opponents were using 4-ply or more. The skill refinement (increased use of the experience base and therefore less time spent in search) was indicated by the number of times a decision was based on the experience base. The growth of the experience base was measured in terms of the number of nodes in the experience base after the end of each game.

Figure I illustrates the results of an experiment in which GINA played against a standard min-max type opponent using a five ply lookahead. These results are typical of all of the experiments pitting GINA against a single opponent. As can be seen, GINA quickly learns the opponents weaknesses and the learning asymptotes. At this point, GINA is winning the vast majority of games by an average of ten or more points.

We followed this with a second set of experiments in which GINA plays against three opponents in a round robin tournament. The idea was to prevent GINA from homing in too quickly on the weaknesses of any one opponent and to see if experiences from one style opponent would negatively effect GINA's performance against others. Figure II illustrates the results of an experiment in which two opponents are min-max type opponents using a four ply lookahead, and the third

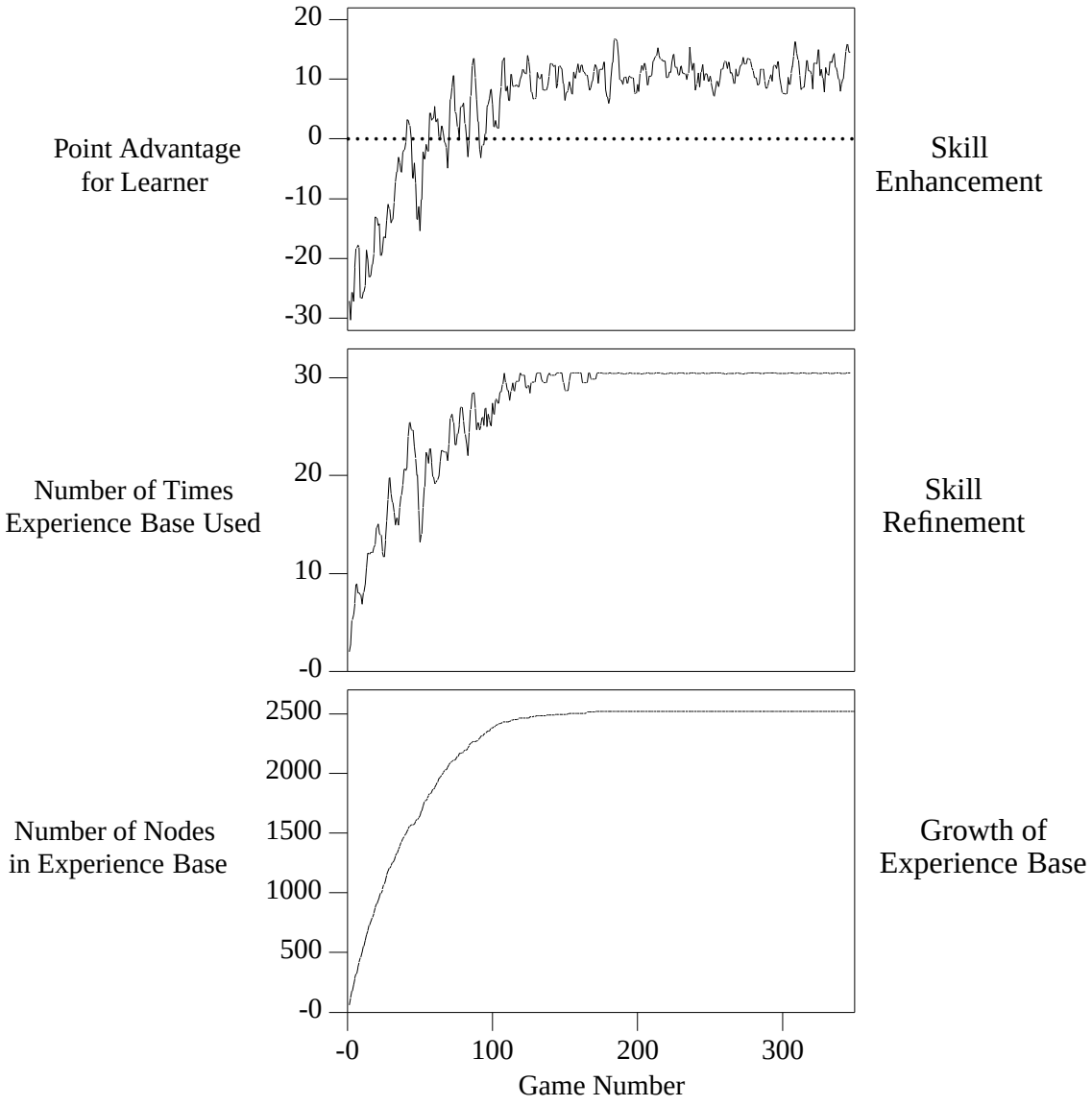

Figure I: Learner vs. Minmax1

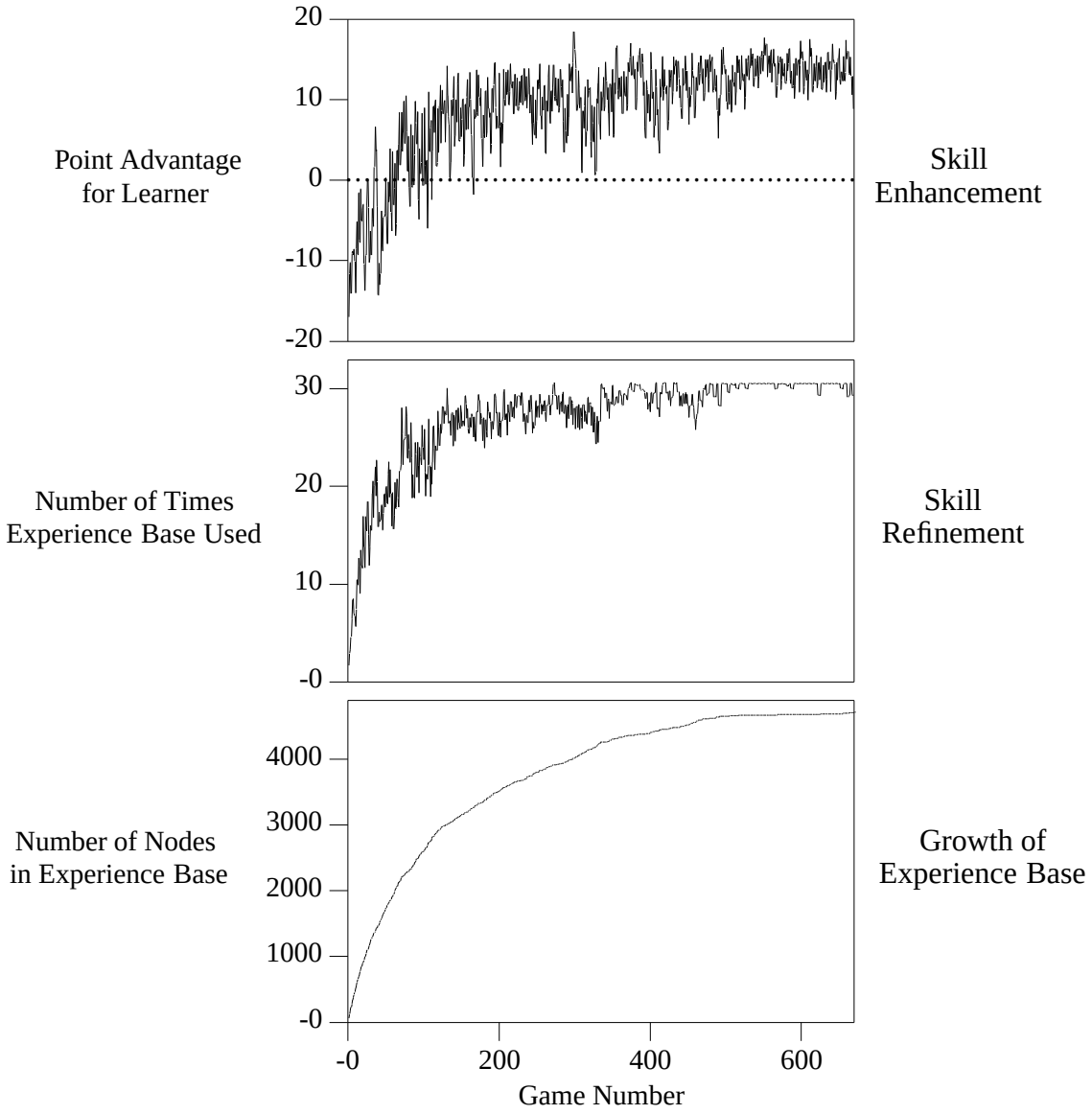

Figure II: Learner vs. Three Opponents (round robin)

opponent has access to a static copy of an experience base from the previous experiment. The results are typical of all the round robin experiments, in that GINA's performance improves drastically, eventually eliminating the need to perform any state-space search. Note that with the added variety of opponents, the experience base takes longer to asymptote and that the experience base contains more information.

4.3. Discussion of Results

We are somewhat surprised and quite pleased at how effective this form of experience-based learning can be in a game playing environment. In every case we observed significant improvements in the quality of play and dramatic decreases in the cpu time required to select moves without uncontrolled growth of the size of the experience base.

At the same time it is important to temper these results with the observation that all of GINA's opponents had one important feature in common: they were all non-adaptive in the sense that they made no attempt to change their behavior in an attempt to improve their performance against GINA. This unchanging behavior undoubtedly enhanced GINA's ability to home in on and exploit weaknesses.

5. FUTURE RESEARCH

We are already in the process of exploring some of the possibilities mentioned earlier, namely, the effects of other ''personalities'' and less predictable opponents. In addition, we are eager to apply these ideas to some fault isolation expert systems whose non-adaptive behavior triggered these ideas in the first place.

6. REFERENCES

1. Kenneth A. De Jong, ''Knowledge Acquisition for Fault Isolation Expert Systems,'' in *Proc. of the Knowledge Acquisition for Knowledge-Based Systems Workshop (Banff, Canada, November 2-7, 1986)*, AAAI (1986).
2. T. Mitchell, R. Keller, and S. Kedar-Cabelli, ''Explanation-Based Generalization: A Unifying View,'' *Machine Learning* **1**(1) pp. 47-80 (1986).
3. R. S. Michalski and R. E. Stepp, ''Learning from Observation: Conceptual Clustering,'' pp. 331-363 in *Machine Learning I*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell,Tioga, Palo Alto (1983).
4. J. Ross Quinlan, ''Learning Efficient Classification Procedures and their Application to Chess End Games,'' pp. 463-482 in *Machine Learning I*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell,Tioga, Palo Alto (1983).
5. A. L. Samuel, ''Some Studies in Machine Learning Using the Game of Checkers, II - Recent Progress,'' *IBM Journal of Research and Development* **11** pp. 601-617 (1967).
6. J. Laird, P. Rosenbloom, and A. Newell, ''Chunking in Soar: The Anatomy of a General Learning Mechanism,'' *Machine Learning Journal* **1**(1) pp. 11-46 (1986).
7. Tom M. Mitchell, Paul E. Utgoff, and Ranan Banerji, ''Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics,'' pp. 163-190 in *Machine Learning I*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell,Tioga, Palo Alto (1983).
8. D. Michie, '''Memo' Functions and Machine Learning,'' *Nature* **218** pp. 19-22 (1968).
9. D. Marsh, ''Memo Functions, the Graph Traversal, and a Simple Control Situation,'' pp. 281-318 in *Machine Intelligence 5*, ed. B. Meltzer and D. Michie,American Elsevier, New York (1970).
10. A. L. Samuel, ''Some Studies in Machine Learning Using the Game of Checkers,'' *IBM Journal of Research and Development* **3** pp. 210-229 (1959).
11. J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor (1975).