GP-Gammon: Genetically Programming Backgammon Players

YANIV AZARIA azariaya@cs.bgu.ac.il
MOSHE SIPPER sipper@cs.bgu.ac.il
Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel

Submitted September 17, 2004; Revised January 10, 2005
Published online: 12 August 2005

Abstract. We apply genetic programming to the evolution of strategies for playing the game of backgammon. We explore two different strategies of learning: using a fixed external opponent as teacher, and letting the individuals play against each other. We conclude that the second approach is better and leads to excellent results: Pitted in a 1000-game tournament against a standard benchmark player—Pubeval—our best evolved program wins 62.4% of the games, the highest result to date. Moreover, several other evolved programs attain win percentages not far behind the champion, evidencing the repeatability of our approach.

Keywords: genetic programming, backgammon, self-learning

1. Introduction

Games, long considered epitomic of human intelligence, have attracted many a researcher in artificial intelligence, ever since the field’s prehistoric times (namely, the 1950s). Tic-tac-toe, checkers, chess, robotic soccer, and multifarious other games have been targeted by those wishing to study (and possibly enhance) machine intelligence. After all, what better proof of the latter than a machine beating us (literally) at our own game?

Specifically, board games such as checkers, Othello, and backgammon have all yielded to machine-learning techniques in the past decades. The basic rules are few and relatively easy to learn, however, excelling at the game is an altogether different matter. An ideal strategy—one that always wins—is usually impossible to obtain (either through human or computer design), but heuristics that perform well against human or machine opponents can be found (albeit with much effort). Commercial interests are also at stake since developing an efficient game strategy can readily be turned into a winning product (as evidenced by the multi-billion dollar computer-game industry).

Our research herein focuses on the game of backgammon, which falls into the category of board games that do not yield to exhaustive analysis (and solution), but which yield to heuristic solving, that is, a heuristic strategy that performs very well against human and machine players can be obtained. The probabilistic nature of the game makes it suitable for learning [20]. The application of machine-learning techniques to obtain strong backgammon players has been done both in academia and industry. The best commercial products to date are Jellyfish [3] and TD-Gammon [20]. For these, suitable interfaces for benchmarking are unavailable, and there are no published results concerning their performance against other
programs. Our benchmark competitor will thus be the freely available Pubeval (described below)—which has become a standard yardstick used by those applying AI techniques to backgammon.

The majority of learning software for backgammon is based on artificial neural networks, which usually receive as input the board configuration and produce as output the suggested best next move. The main problem lies with the network’s fixed topology: The designer must usually decide upon this \textit{a priori}, whereupon only the internal synaptic weights change. (Nowadays, one sometimes uses evolutionary techniques to evolve the topology \cite{21}).

The learning technique we have chosen to apply is \textit{Genetic Programming} (GP), by which computer programs can be evolved \cite{7}. A prime advantage of GP over artificial neural networks is the automatic development of structure, i.e., the program’s “topology” need not be fixed in advance. In GP we start with an initial set of general- and domain-specific features, and then let evolution determine (evolve) the structure of the calculation (in our case, a backgammon-playing strategy). In addition, GP readily affords the easy addition of control structures such as conditional statements, which may also evolve automatically.

This paper details the evolution of highly successful backgammon players \textit{via} genetic programming. In the next section we present previous work on machine-learning approaches to backgammon along with a few examples of applications of GP to other games. In Section 3 we present our algorithm for evolving backgammon-playing strategies using genetic programming, with the presence of an external opponent as “teacher.” Section 4 presents the self-learning approach to the problem, and in Section 5 we compare the two approaches. This is followed by Section 6 that discusses the evolved strategies. Finally, we present concluding remarks and future work in Section 7.

2. Previous work

In 1989, Tesauro presented Neurogammon \cite{18}, a neural-network player evolved using supervised learning and several hand-crafted input features of the backgammon game. This work eventually led to TD-Gammon, one of the top two commercial products to date \cite{20} (Section 1). This work is based on the Temporal Difference (TD) method, used to train a neural network through a self-playing model—i.e., learning is accomplished by neural networks playing against themselves and thus improving.

In 1997, Pollack et al. \cite{11} presented HC-Gammon, a much simpler Hill-Climbing algorithm that also uses neural networks. Under their model the current network is declared ‘Champion,’ and by adding Gaussian noise to the biases of this champion network a ‘Challenger’ is created. The Champion and the Challenger then engage in a short tournament of backgammon: if the Challenger outperforms the Champion, small changes are made to the Champion biases in the direction of the Challenger biases.

Another interesting work is that of Sanner et al. \cite{15}, whose approach is based on cognition (specifically, on the ACT-R theory of cognition \cite{1}). Rather than trying to analyze the exact board state, they defined a representational abstraction of the domain, consisting of general backgammon features such as blocking, exposing, and attacking. They maintain a database of feature neighborhoods, recording the statistics of winning and losing for each such neighborhood. All possible moves are encoded as sets of the above features; then, the move with the highest win probability (according to the record obtained so far) is selected.