01 Jan 2001

A Parallel Computer-Go Player, using HDP Method

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Xindi Cai

A parallel computer-Go player, using HDP method

Xindi Cai Donald C. Wunsch II

Applied Computational Intelligence Laboratory
Dept. of Electrical and Computer Engineering
University of Missouri - Rolla
Rolla, MO 65409-0249 USA
cai@umr.edu dwunsch@ece.umr.edu

Abstract:

The game Go has simple rules to learn but requires complex strategies to play well, and, the conventional tree search algorithm for computer games is not suited for Go program. Thus, the game Go is an ideal problem domain for machine learning algorithms. This paper examines the performance of a 19x19 computer Go player, using heuristic dynamic programming (HDP) and parallel Alpha-Beta search. The neural network based Go player learns good Go evaluation functions and wins about 30% of the games in a test series on 19x19 board.

1 Introduction

Go is a deterministic, perfect information, zero-sum game of strategy between two players. Players take turns to place black and write pieces (called stones) on the intersections of the lines in a 19x19 grid, called the Go board. Once played, a stone cannot be removed, unless captured by the other player. To win the game, each player seeks to surround more territory (empty grids) by one's own stones than the opponent. Adjacent stones of the same color form strings, and hence groups; an empty intersection adjacent to a stone, a string, etc, is called its liberty. A group is captured when its last liberty is occupied by the opponent's stone. To prevent loop, it is illegal to make moves that recreate prior board positions (rule of Ko). A player can pass any time. The game ends when both players pass in consecutive turns. There are excellent books available on the game of Go [1].

The conventional tree search algorithms make poor Go programs. The reasons why the brute-force search is not efficient for Go are: (1) the possible moves at each position (i.e., the branches of one node of the search tree) is a huge number, let along the legal moves; (2) many situations, especially in mid-game, in Go require a very deep looking in order to lead; (3) the formulation of Go evaluation function, which is not explicit, is very difficult.

The heuristic dynamic programming (HDP) has been used to train neural networks for learning game evaluation functions [2], [3], [4]. The HDP method is an elegant way of doing reinforcement learning because: (1) it uses the environment as its own model, and (2) it proposes to use a neural networks of the form f(s, w), where w is the adjustable weight vector, to approximate the value function, $f^{\pi}(s)$. Instead of storing a separate value function for each state, learning is achieved by adjusting the weights to minimize the mean squared error between f(s, w) and $f^{\pi}(s)$. The HDP method is an incremental learning procedure specialized for prediction problems where the inputs are applied in sequence [5]. The algorithm adjusts the weights as follows:

$$\Delta w_t = \alpha (P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \Delta_w P_k \quad (1)$$

and it minimizes the following criterion function:

$$J(w) = \sum_{p=1}^{P} \sum_{k=1}^{N_p} \lambda^{N_p - k} (zN_p - G(x_p(k)))^2 \quad (2)$$

In the above equations, P is the number of examples, e.g., the number of games; N_p is the number of steps in the p^{th} example, which is unknown until the final outcome is determined; zN_p is the final output (determined by the game rule) of the p^{th} example at the end of the game p. Game p consists of a series of states $x_p(k)$, k=1, 2, .., N_p. G($x_p(k)$) is the output of the network when $x_p(k)$ is presented; and, λ, between 0 and 1, is a parameter which is used to place more emphasis on predictions temporally close to the outcome.

One of the key determinants of a game playing program's strength is the depth of the game tree search. Therefore, parallelism is used to search deeper trees in the same amount of real time. Tree decomposition algorithms extract

0-7803-7044-9/01/$10.00 ©2001 IEEE

parallelism by creating split nodes, where the subtrees rooted at the node are searched concurrently.

The Principle Variation Splitting (PVS) is a tree decomposition algorithm for a depth-first Alpha-Beta search [6]. It creates parallel work by splitting nodes along the principal variation. Recurring down the principal variation concentrates the parallel effort and backing up the principal variation carries important window information for the search of the next subtrees. The philosophy of the PVS is to back up a score as quickly as possible to each splitting node so that this score can be used by all processors when they go their own way searching the remaining moves at the node.

The PVS works as follows: Just as in single-processor systems, the PVS carries out a sequence of iteratively deepening searches. Let the principal continuation of moves found on the (n-1)st iteration be denoted by $m_1, m_2, ..., m_{n-1}$ and the nodes (corresponding to game positions) on this continuation be denoted by $V_0, V_1, V_2, ..., V_{n-1}$ where V_0 is the root of the tree and V_{n-1} corresponds to the position reached at the end of the continuation. On the nth iteration, all processors initially search this continuation down to V_{n-1} and then dynamically divide up search for the moves rooted there. When search of these moves is complete, a score is back up to V_{n-1}. Then, the remaining moves rooted at V_{n-2} are dynamically divided up and searched. When search of these moves is complete, a score is back up to V_{n-2}, and so on. This process is repeated at each splitting node, until finally, moves at the root are dynamically divided up and searched and a new score and new principal continuation are determined.

As mentioned above, HDP is good at learning the implicit evaluation function by using the environment as the model, and PVS can give us more power to have a deeper look in Go. In this paper, we try to combine HDP and PVS in playing 19x19 Go and report some results of the new idea implemented utilizing the experience of [7].

2 Network Architecture

The network architecture is shown on Figure 1. Our Go player contains four systems, i.e., likely move generator, big move prediction, critic net evaluation and parallel Alpha-Beta search. The likely move generator uses neural networks as the learning structure and is trained on high quality games played by human masters. The target is to select 30 - 40 plausible moves at any board position. Together, some examples of "bad" moves are also provided for fast learning. The big move prediction module identifies moves that affect the safety of groups, for either player. A group consists of several strings, which are not directly connected, but have a close relationship. For both sides, the potential safety and connectivity of groups will greatly affect the final result of the game, and hence determine the next move, or a series of moves, which we may treat as a strategy. The big move prediction offers some directions for the likely move generator to consider. The critic net evaluation is composed two subsystems, territory control prediction unit and string safety prediction unit, which use neural networks to predict territory measure of each empty board intersection and safety possibility of each string, respectively. Thus evaluation function, according to their outputs, can assign credits on the possible moves generated. The parallel alpha-Beta search does the job of finding a best path from the current board to certain depths afterwards. The credits on the leaf nodes are the outputs of the evaluation function, after predicting those different boards. Wally [8], a weak public domain program (rating ~30 Kyu), is served as the opponent, providing the BLACK moves.

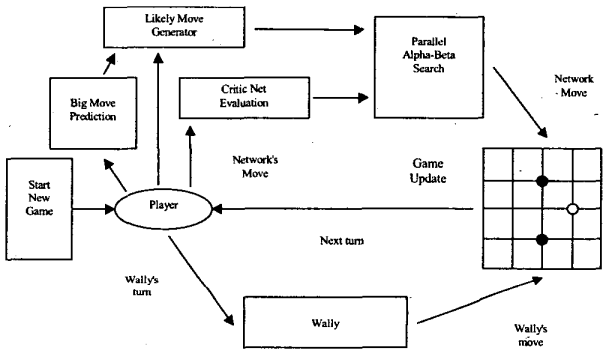

Figure 1: Block diagram of Go player network

3 Networks Training and Results

For each intersection on the board, seven components, representing influence value for empty point, No. of stones and liberties in the string, if occupied, for both sides, are created. A 7x7 diamond window, which reads 25 board intersections per prediction, has 25x7 items in the input vector. This determines the neurons in the input layer roughly. In big move prediction, influence values are used to combine strings into groups, determine potential group eye space and mark crucial stones. The two subsystems, namely territory control unit and string safety unit, in the critic network are trained separately on different area of the board, i.e., corner, side and center, to construct three convolutional networks each. However, there is only one neural network for likely move generator system. After training on 100 games, the performance of each network is

listed on Table 1, 2 and the likely move generator system can guess the plausible moves 72% of the time, considering the top 20 moves. Also, the speedup of the parallel Alpha-Beta search, based on a game tree of depth 8 and width 16 is listed on table 3. Finally, the Go-player is tested with 100 games and wins about 30 of them.

Table 1: Rate of correct territory control prediction on different areas of board and for different players

	corner	side	center	black	white
Territory Control Unit	82%	92%	81%	94%	91%

Table 2: Rate of correct string safety prediction on different areas of board, and for live and dead

	corner	side	center	live	dead
String Safety Unit	56%	78%	81%	80%	79%

Table 3: Speedup of PVS

processor number	2	4	8	16
speedup	1.8	3.0	4.1	5.0

4 Conclusion

The results show that our computer Go player can learn to play from zero knowledge. This simple Go engine does not act as a strong Go player, but it demonstrates that the principle of dynamic programming can be utilized to in corporate machine learning in playing the game Go. Adding more Go-related knowledge will surely improve the performance, in forming a meaningful J function and predicting it accurately.

References

[1]. Arthur Smith, *The game of Go*, Charles Tuttle Co. Tokyo, Japan, 1956.

[2]. G. Tesauro, "Practical Issues in temporal difference learning", *Machine Learning*, No. 8, pp. 257-278, 1992.

[3]. R. Zaman, D. Prokhorov and D. Wunsch, "Adeptive Critic Design in Learning to play Game of Go", *Proc. of ICNN*, Houston, Vol. 1, pp. 1-4, 1997.

[4]. N. N. Schraudolph, P. Dyan and T. J. Sejnowski, "Temporal learning of position evaluation in the game of Go", *Advances in Neural Information Processing*, Vol. 6, pp.817-824, 1994.

[5]. S. Sutton, "Learning to predict by the method of temporal differences", *Machine Learning*, No. 3, pp. 9-44, 1988.

[6]. M. Newborn, "Unsynchronized iteratively deepening parallel Alpha-Beta search", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 5, pp. 687-694, 1988.

[7]. R. Zaman, Applications of neural networks in Computer Go, Ph.D. dissertation, Texas Tech University 1998.

[8]. B. Newman, "Wally - a simple minded Go-program", shareware Go program available by anonymous ftp from ftp://imageek.york.cuny.edu/nngs/Go/comp/.