Proceedings of the Twenty-Ninth International
Florida Artificial Intelligence Research Society Conference

Go-Ahead: Improving Prior Knowledge Heuristics by Using Information Retrieved from Play Out Simulations

Gabriel Machado Santos
Computer Science Department
Federal University of Uberlandia - UFU
Uberlandia, Brazil
gabrielmsantos@gmail.com

Rita Maria Silva Julia
Computer Science Department
Federal University of Uberlandia - UFU
Uberlandia, Brazil
ritasilvajulia@gmail.com

Abstract

The proposal behind this paper is the introduction of a new agent denominated Go-Ahead: this is an automatic Go player that uses a new technique in order to improve the accuracy of the pre estimated values of the moves that are candidates to be introduced into the classical Monte Carlo tree search (MCTS) algorithm which is used by many of the current top agents for Go. Go-Ahead is built upon the framework of one of these agents: the well known open source automatic player Fuego, in which these pre estimated values are obtained by means of a heuristic called *prior knowledge*. Go-Ahead copes with the task of refining the calculations of these values through a new technique that performs a balanced combination between the *prior knowledge* heuristic and some relevant information retrieved from the numerous *play out* simulation phases that are repeatedly executed throughout the Monte Carlo search. With such a strategy, Go-Ahead provides the contribution of enhancing the MCTS process of choosing appropriate moves. Further, this new approach attenuates the supervision level inherent to this process due to the following fact: it allows for the lessening of the impact of the prior knowledge heuristics through strengthening the impact of play out information. The results obtained in tournaments against Fuego confirm the benefits and the contributions provided by this approach.

Introduction

This paper presents an agent player for the game of Go named Go-Ahead. The main motivation here lies in the fact that the technical and theoretical complexity inherent to the task of building well performing agents for Go is very similar to that required in the construction of agents that are able to deal with several important everyday problems of real life (Russell and Norvig 1995). Therefore, the complexity of the game is not derived from the quantity of rules but rather from the diversity of game situations provided by the extremely large state space and branching factor, as shown in Table 1.

Fuego is built upon an open source framework with the same name. It is also widely used to support the development of many automatic players for Go (Enzenberger et al. 2010), including the agent proposed in this paper. The search for the best move in the Fuego player is performed by means of the Monte Carlo Tree Search (**MCTS**) algorithm (Chaslot 2010). Each game simulation is designated as an *episode* and it is divided into four distinct phases: *selection*, *expansion*, *play out* and *back propagation*.

Table 1: Average branching factor and space state.

Board Game	Average branching factor	Log(space state)
Connect Four	4	13
Draughts (checkers)	2.8	18
Backgammon	250	20
Chess	35	50
Go (19x19)	250	172

The contribution of the agent Go-Ahead consists of creating an algorithm capable of extracting useful information related to the nodes (moves) from the *play out* phase, and to use such information in order to increase the accuracy of the pre estimated values of the nodes that are candidate to be inserted into the search tree during the *expansion* phase.

This information refers to two main parameters: the simulation frequency of each move in the *play out* phase and the reinforcements (defeat or victory) that are obtained at the end of these *play outs*. This information will be progressively stored in a hash table while the simulation process goes on. By performing a balanced combination between the values retrieved from this hash table and the prior knowledge values used by Fuego, Go-Ahead provides two distinct contributions: first, it increases the value accuracy of the nodes which are candidates to be inserted into the search tree during the *expansion* phase, which enables the agent to enhance the process of choosing appropriate moves. Second, the balancing in the combination of these values (obtained by means of an adjustable parameter) represents an interesting alternative to attenuate the supervised character of the calculations of the node evaluations in Fuego, since it allows for a reduction in the impact of the prior knowledge heuristic by strengthening the impact of the knowledge (information) retrieved from the search process. The auspicious results obtained by Go Ahead in tournaments against Fuego proves that the approach proposed in this paper represents an appropriate strategy for attenuating the supervision and for improving the performance in agents for Go.

This paper is organized as follows: section "Theoretical Foundations" presents the core concepts for the understanding of this paper. Section "Go-Ahead" describes the techniques used in the Go-Ahead player. Section "Experiments and Results" shows the experimental results obtained. Finally, the final considerations and future work planning are presented in "Conclusions and Future Work".

Theoretical Foundations

For purpose of objectivity, the theoretical foundations and the techniques concerning the present work are presented according to the way in which the MCTS based agents for Go make use of them.

The Game of Go

Go is a territorial game whose board is usually set up with 19 vertical and 19 horizontal lines. It can be thought of as a piece of land to be shared between the two players (corresponding to the black and the white stones). The game starts with an empty board and the players take turns to place the stones on an empty intersection.

The main objective of the player is to conquer the largest territory possible by surrounding the maximum area of the board with its corresponding stones. The following concepts are inherent to Go: a **group** is a connected set of stones and the term **liberty** refers to an empty intersection adjacent to a group. In terms of actions, a group must be captured (removed from the board) whenever it has lost all its liberties.

Monte Carlo Tree Search

The MCTS algorithm performs the search for the best move by means of a game tree built through MC simulations. This process is guided by two distinct policies, named tree policy (or $\pi_{in-tree}$) and play out policy (or $\pi_{simulation}$). Each tree node represents a move. Its structure is comprised of at least two pieces of information: *a*) The move value v_i: represents the average evaluation value obtained by the node in the games that have been simulated; *b*) the number n_i: indicates the number of simulations in which the node has been involved. A board state represents the exact configuration of the board game at a specific moment.

The search tree is built through an iterative process in which an arbitrary number N of games is simulated. Each game simulation is named here as an **episode** and corresponds to a complete path on the search tree defined from the root (current state) up to the leaf (endgame state). Each episode consists of four distinct phases: *a*) The **selection**; *b*) The **expansion**; *c*) The **play out simulation**; and *d*) The **back propagation**.

The **selection** phase consists of following down, from the root node of the search tree, the path that is defined by a tree policy $\pi_{in-tree}$ which always selects as next node that which has the the best value. This process continues, recursively, up to the current leaf of the search tree, in which a new leaf node, defined by the *prior knowledge* heuristics, will be inserted. The insertion of a new leaf node defines the **expansion** phase. From this point on, in the play out phase, the path of the game simulation is defined by a set of rules, called $\pi_{simulation}$ **policy**, up to an endgame state. After that, a reinforcement representing victory, defeat or draw is passed to the search algorithm, which uses it to update the nodes involved in the selection and expansion phases of the current episode (called *MCTS updating process*). This updating process corresponds to the **back propagation** phase. As soon as the N episode iterations are concluded, the MCTS algorithm is able to point out the best move to be executed from the current board configuration.

AMAF: An Alternative to the Standard MCTS Updating Process

As shown in the previous section, in the MCTS Updating Process, at the end of each episode, the updating value of a certain move is computed so as not to be affected by other moves on the board or in subsequent turns. Noticeably, the All Move As First (AMAF) (Gelly and Silver 2011) updating technique considers the general value of a move regardless of when it is played. The *all moves as first* value $\tilde{Q}(s,a)$ is the mean outcome of all simulations in which action a is selected *at any turn* after s is found, as shown in 1.

$$\tilde{Q}(s,a) \leftarrow 1/N(s,a) \times \sum_{i=1}^{N(s)} \tilde{I}_i(s,a) z_i \qquad (1)$$

In the update rule 1, $N(s)$ is the total number of visits to the game state s regardless of the previous actions that have been simulated; and $N(s,a)$ is the total number of simulations in which move a has been chosen from state s, $\tilde{I}_i(s,a)$ is an indicator function returning 1 if the state s has been found at any step t of the ith episode, and action a was selected at any step $u \geq t$, or 0 otherwise; z_i is the outcome of the ith episode. It is important to note that both, Black and White moves, are regarded to be distinct actions, even if they are played at the same intersection.

Tree Policies

This section presents the main $\pi_{in-tree}$ policies that are used in the current MCTS based agents for Go.

UCT The UCT, proposed in 2006, is a method to define the path in the selection phase. Basically, it corresponds to a particular application of the Upper Confidence Bounds (UCB) technique (Auer, Cesa-Bianchi, and Fischer 2002) to tree structures, that is, it combines the UCB strategy along with a search tree algorithm. The main idea behind this strategy is to use the UCB method in each step of the selection, always choosing the node which maximizes the following expression 2:

$$Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} \qquad (2)$$

where the $Q(s,a)$ value represents the average number of victories that have been obtained in the game simulations by taking the action a from the game state s, in which the game state is just a representation of the game at a specific moment; c is an exploration constant and the meaning of $N(s)$ and $N(s,a)$ has already been presented in the update rule 1.

The main advantage of this strategy is the balance between the exploration (a tendency to explore new regions of the search tree) and exploitation (a tendency to keep exploring favorable regions of the search tree) during the search process.

RAVE The RAVE algorithm modifies the MCTS approach by introducing the AMAF heuristic, which increases the knowledge extracted from a play out at the cost of including knowledge that may be biased or less relevant (Brügmann 1993). By combining the MCTS algorithm with the AMAF heuristic, RAVE allows the information to be shared between the sub trees of the search tree during the process. The RAVE strategy can be represented by the update rules 3 and 4:

$$m(s_t, a_x) \leftarrow m(s_t, a_x) + 1 \tag{3}$$

$$\tilde{Q}(s_t, a_x) \leftarrow \tilde{Q}(s_t, a_x) + 1/m(s_t, a_x)[R_t - \tilde{Q}(s_t, a_x)] \tag{4}$$

where $\tilde{Q}(s, a)$ is the AMAF value of an action $a \in A(s)$; $A(s)$ represents the set of legal moves available in the state s; s_t is the state s selected at time t of a i-th episode; a_x is the action $a \in A(s_t)$ selected at time x of the same i-th episode, with $x \geq t$; $m(s, a)$ represents the number of times that action a was selected in any subsequent state to s; and, finally, R_t is the outcome returned at the end of the last simulated play out.

Go-Ahead

This section presents the system Go-Ahead: an agent for Go that enhances the performance and attenuates the supervised character of the remarkable automatic player Fuego by using statistical information (Wasserman 2004) retrieved from the numerous *play out* phases of the current MC search. More specifically, in the pre expansion phase of Go-Ahead, this information, which is stored in a hash table, will be combined with the prior knowledge heuristic in order to increase the accuracy of the move estimation and to provide more autonomy to the agent (since it attenuates the impact of this heuristic).

Search Architecture

The Dynamic Estimator Module (DEM) copes with the following tasks: to update the hash table values to the extent that the *play outs* are simulated; and to calculate the move evaluations by combining these values with the prior knowledge heuristic in the *pre expansion* phase. These dynamics can be resumed as follows:

1. After the selection phase, all candidate moves (children of the last selected node) are assigned with the pre estimated value calculated by the DEM;
2. The candidate move with maximum value will be added to the search tree as a new leaf (expansion phase);
3. The play out simulation phase is fired off from the new leaf inserted at the last expansion phase;
4. As soon as each play out phase is concluded, the set composed of the moves that have been simulated in this play out, as well as the reinforcement obtained (victory or defeat), are passed as new information to the DEM. This module uses this information to update the values corresponding to these moves in the hash table.
5. The back propagation phase is triggered, so as to update the values of the tree nodes;
6. The whole of the search process is triggered again (limited to the number N of episodes that were previously established for each search process).

The Dynamic Estimator Module

This section presents the DEM in greater detail.

DEM node structure The DEM is basically composed of a hash table that stores nodes containing useful information about play out simulated moves. Each node within the DEM is composed of three variables: the $move\_number$, which is an integer representing the hash key corresponding to the move itself. It corresponds to the position occupied by a stone on the board game. It is calculated through a function $Position(x, y)$ that returns a unique value for each (x, y) coordinate. For example, the coordinate $(5, 5)$ (line 5 and column 5) is represented by the integer 105, the coordinate $(10, 3)$ (line 10 and column 3) is represented by the integer 203 and so on; $move\_counter$, which is an integer that indicates how many times this move has been simulated in the current game; and finally, a $move\_value$ that is a double representing the rate of victories accumulated at the play outs of the current game in which this move was simulated.

Updating the Hash Table nodes At the end of each play out phase, the set S composed of the moves that have been simulated in this phase and the resulting reinforcement (*R*) are passed to the DEM. This module then proceeds in the following way: it firstly updates (recalculates) the data related to the elements of S that are present in the hash table; next, the DEM calculates the data related to the elements of S that are not present in the hash table and insert these into it. Both calculations (used to update or to include a new datum in the hash table) are performed according to the rules 5 (which updates/includes the datum *move_value* $M_{(a)}$ of a move *a*) and 6 (which updates/includes *move_counter* $C_{(a)}$ of this particular move *a*).

$$M_{(a)} \leftarrow \frac{(M_{(a)} \times C_{(a)}) + R}{C_{(a)} + 1} \tag{5}$$

$$C_{(a)} \leftarrow C_{(a)} + 1 \tag{6}$$

Combining Prior Knowledge with the DEM Estimation In the classical MCTS algorithm, each node that is a candidate to be inserted into the MC tree in the *expansion* phase is assigned with a prior knowledge value (Gelly et al. 2006). As this value is obtained through some heuristic knowledge and through a set of empirically computed data, it presents a static characteristic, a fact that makes it not very accurate. In order to deal with this shortcoming, the approach presented in this paper tries to refine the calculations of these nodes in the following way: it combines the prior knowledge heuristic with the dynamic information retrieved from the history of the play out move simulations performed by the MCTS algorithm (stored in the hash table), as shown in the estimation rule 7. By using this procedure, besides improving the process of choosing an appropriate move by means of a more accurate node evaluation, Go-Ahead also provides a bit more autonomy to the MCTS based agents, since it uses the hash table information to attenuate the impact caused by the use of the prior knowledge heuristics.

$$M_{PE(a)} \leftarrow (\gamma \times PK_{(a)}) + ((1-\gamma) \times M_{(a)}) \quad (7)$$

In the estimation rule 7: $M_{PE(a)}$ is the current pre estimation value calculated by Go-Ahead for a node *a* that is candidate to be inserted into the MC tree in the *expansion* phase; $PK_{(a)}$ is the prior knowledge value associated to this move a; $M_{(a)}$ is the hash table value for move a; γ is a constant that weights the prior knowledge and the hash table values of a.

Experiments and Results

This section presents all the test scenarios performed in order to evaluate the performance of the agent Go-Ahead in tournaments against the version 1.1 of Fuego.

In these tournaments, the value adopted for the weighting constant γ in rule 7 is equal to 0.8, since Go-Ahead, playing with such configuration in empirical tests, obtained the most enhanced performance. It means that Go-Ahead plays with an autonomy level of about 20% superiority over that of its opponent Fuego. In future works, the authors intent to investigate if any benefit can result from varying this constant in different stages of the game.

For completeness purposes, in the evaluative tournaments Go-Ahead plays either as black player, or as white player. The tests are executed within scenarios I, II and III, involving 8000, 16000 and 64000 episodes, respectively. The performance of the agent is estimated in terms of its victory rate in the tournaments. The processor used is an Intel Core 2 Quad 2.4 GHz with 8GB of RAM. The games are executed in two distinct board configurations: 9x9 and 19x19. The results for scenarios I, II and III are shown in table 2. Each line of the table represents a tournament composed of 500 games.

Table 2 shows that the victory rates obtained by Go-ahead playing against Fuego in 9x9 game board in scenarios I, II, III were 62%, 56% and 59%, respectively. On the other hand, its victory rates in 19x19 game board, in the same scenarios, were 58%, 54% and 52%, respectively.

These results confirm that the approach adopted in Go-Ahead, besides allowing for a greater autonomy of the agent, really improves its performance, even in the very hard situations represented by 19 X 19 game board configurations.

Although the search runtime has been increased by 9% in the process, the authors consider that this shortcoming is satisfactorily compensated by the gains that the approach brought to the automatic player.

Conclusions and Future Work

This paper presented Go-Ahead, an agent for Go that uses information retrieved from the play out simulations to increase the accuracy of the prior knowledge heuristics used by Fuego.

Table 2: Win Rate of Go-Ahead x Fuego in Test Scenario I, II and III

Board Size	Test Scenario	**Win Rate of Go-Ahead**
9x9	I	62%
19x19	I	58%
9x9	II	56%
19x19	II	54%
9x9	III	59%
19x19	III	52%

Evaluative tournaments involving Go-Ahead and Fuego confirmed that this strategy, besides providing more autonomy to the MCTS based automatic players, make them able to perform a more accurate move estimation - a factor that increases their performance in matches.

In Fuego, whenever there is a set of candidate nodes to be inserted into the search tree (in the *expansion* phase) that presents the same prior knowledge value, the agent makes the choice in a random way. In a distinct way, Go-Ahead, in the same situation, is able to choose the move that, considering the history of the previous simulations, presents a higher level of quality.

It is also interesting to point out that, in the evaluative tournaments, both players sometimes naively executed some so called *bad moves*, like playing on the first line when not required. In this sense, in future works the authors intend to investigate appropriate heuristics to cope with these flaws. Furthermore, The authors intent to investigate if any benefit can result from updating the prior knowledge with new data for future rounds.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Brügmann, B. 1993. Monte-Carlo Go. Technical report, Citeseer.

Chaslot, G. 2010. *Monte-Carlo tree search*. Ph.D. Dissertation, PhD thesis, Maastricht University.

Enzenberger, M.; Muller, M.; Arneson, B.; and Segal, R. 2010. Fuegoan open-source framework for board games and Go engine based on Monte-Carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on* 2(4):259–270.

Gelly, S., and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence* 175(11):1856–1875.

Gelly, S.; Wang, Y.; Munos, R.; and Teytaud, O. 2006. Modification of UCT with Patterns in Monte-Carlo Go.

Russell, S., and Norvig, P. 1995. Artificial intelligence: A new approach.

Wasserman, L. 2004. *All of statistics: a concise course in statistical inference*. Springer.