BTT-Go: An Agent for Go That Uses a Transposition Table to Reduce the Simulations and the Supervision in the Monte-Carlo Tree Search

Eldane Vieira Junior and **Rita Maria Silva Julia**
Department of Computer Science
Federal University of Uberlandia - UFU
Uberlandia, Brazil
eldanevieira@gmail.com, rita@ufu.br

Abstract

This paper presents BTT-Go: an agent for Go whose architecture is based on the well-known agent Fuego, that is, its search process for the best move is based on simulations of games performed by means of Monte- Carlo Tree Search (MCTS). In Fuego, these simulations are guided by supervised heuristics called *prior knowledge* and *play-out policy*. In this context, the goal behind the BTT-Go proposal is to reduce the supervised character of Fuego, granting it more autonomy. To cope with this task, the BTT-Go counts on a Transposition Table (TT) whose role is not to waste the history of the nodes that have already been explored throughout the game. By this way, the agent proposed here reduces the supervised character of Fuego by replacing, whenever possible, the *prior knowledge* and the *play-out policy* with the information retrieved from the TT. Several evaluative tournaments involving BTT-Go and Fuego confirm that the former obtains satisfactory results in its purpose of attenuating the supervision in Fuego without losing its competitiveness, even in 19x19 game-boards.

1 Introduction

Games have been one of the main areas of research due to the fact that they are in tune with real life problems, such as those which arise from the interaction of man and machine, as well as deal with unpredictable situations (Brügmann 1993).

The game Go has been a great challenge due to its complexity in terms of strategy compared to other ones, as shown in Table 1 (Campos and Langlois 2003). That is why it is an important field for researchers in the Artificial Intelligence area.

This paper presents the Go player agent BTT-Go, which aims at reducing the supervised character of the player program GO called Fuego (Enzenberger et al. 2010), without compromising its competitiveness. It is interesting to point out that a preliminary theoretical proposals of BTT-Go were introduced by the authors in (Junior and Julia 2013). In the present work, the authors detail the final architecture of the system and the results obtained in evaluative tournaments against other successful agents for Go.

Table 1: Ramification factor and space states of same games.

Game	Branch	States
Chess	30 – 40	10^5
Checkers	8 – 10	10^{17}
Backgammon	±420	10^{20}
Othello	±5	$< 10^{30}$
Go 19x19	±360	10^{160}
Abalone	±80	$< 3^{61}$

The Fuego program uses the search algorithm based on Monte-Carlo (MC) simulations (Brügmann 1993), the algorithm UCT (Upper Confidence Trees) (Coulom 2007b) and Rapid Action Value Estimation (RAVE) (Gelly and Silver 2011), besides using *prior knowledge* and *opening book*, for the realization of the games. Every move is chosen after that the execution of an arbitrary quantity of game simulations has been performed. Each game simulation is comprised of two phases: *tree construction* and *play-out*, which are based on supervised heuristics called *prior knowledge* and *play-out policy*, respectively.

During a game of Go, the same state can appear again in different game sequences (transposition), thus, creating interest in the use of a TT (Childs, Brodeur, and Kocsis 2008) as a repository of states which have been previously evaluated. In this manner, to reach its objective, the BTT-Go uses a TT that allows the agent to store as information the updated values of the nodes which represent the history of the previous evaluations performed over the course of the current game. Whenever such an information is available, it is used to substitute the *prior knowledge* and the *play-out policy* supervisions. In particular, each time that a visited node in the *play-out* phase has its evaluation available on the TT, this value can be retrieved, which allows the interruption of the current game simulation. Obviously, such a reduction on the quantity of simulations also represents a reduction in the supervising of Fuego.

It is important to highlight that other current agents for Go, such as *Migos* (van der Werf, Van Den Herik, and Uiterwijk 2003) and *Crazy Stone* (Enzenberger and Müller 2010), also take advantage of TT, however with a different approach, as summarized in the section 3. The results obtained from evaluation matches against Fuego show that BTT-Go, is able to maintain a good competitiveness against its adversary (average victory rate of 42%, as black player, and of 32%, as white player), even on 19x19 boards, which proves that the way in which it uses TT, as a tool for reducing Fuego supervision heuristics, was extremely successful.

The results also point to a significant reduction in the BTT-Go search time when compared to Fuego, once that it performs fewer simulations. Besides this, the BTT-Go was also tested in matches against Gnu Go (Bump 2003), which shows a much superior performance (average victory rate of 96%, as black player, and 85% as white player).

For the purpose of interpreting such results it is important to remember that the black player tends to have an ample advantage over the white player, once that it executes the first move of the game.

2 Theoretical Foundations

This section presents a short description of the game of Go. Further, it resumes some techniques and tools used in BTT-Go that are very suitable to the game problems.

2.1 The Game of Go

The game of Go is a board game that is disputed between two players (black and white pieces), which was created around 4000 years ago. The objective of the game is to conquer the largest area of board territory possible, along with the capture of your opponent's pieces. Both players try to occupy the *liberties* of a piece or group of pieces, being that *liberties* are free positions adjacent to a piece.

The game of Go can be played on different sized boards, such as 9x9, 13x13 and 19x19. In general this means, the bigger the board, the more complex the game.

2.2 MCTS Search Algorithm

The MCTS search (Chaslot et al. 2006) consists of an algorithm based on a tree. The choice of the best move occurs through a simulation sequence of complete games performed from a current state. Each of these simulations is called an episode. The quantity of episodes for a search is determined in function of a pre-established time.

A search episode for the algorithm MCTS is realized in four steps: *selection*, *expansion*, *play-out* and *back-propagation*, being that *selection* and *expansion* are those steps which compose the so called *tree construction* stage. In the following there is a description of each step realized in an episode of the MCTS algorithm (Gelly and Silver 2011).

- *Selection*: Path defined by the best tree nodes from the root. The evaluation of the nodes is carried out through a strategy known as tree policy. The *selection* phase is only interrupted when a node that does not belong to the tree is encountered.
- *Expansion*: At the end of the *selection*, the best leaf node defined through *selection* is inserted onto the tree.
- *Play-out*: Successive simulations of moves from the node inserted into the *expansion* until it reaches an end game state. Such moves are indicated through heuristics called *play-out policy*.
- *Back-propagation*: Once a game simulation is concluded, the result (victory or defeat) is retro-propagated through the path simulated on the tree and used as a parameter to update the values of the nodes on that path. The general equation that guides the updating of values for a tree node *s'* obtained from the simulation of the execution of an action *a* from the state *s* (then here *s'* is represented by *(s,a)*) is:

$$Q_{(s,a)}^{updated} = Q_{(s,a)}^{current} + Vict_{(s,a)} + Simul_{(s,a)} \qquad (1)$$

onde,

- $Q_{(s,a)}^{updated}$: New value of *s'* after the updating of the *back-propagation*.
- $Q_{(s,a)}^{current}$: Value of *s'* before updating.
- $Vict_{(s,a)}$: Statistics for victories in *play-outs* that pass through *s'* in the current search.
- $Simul_{(s,a)}$: Statistics for *play-outs* that pass through *s'* in the current search.

The adjustment portion is made by the following term of equation 1: $Vict_{(s,a)} + Simul_{(s,a)}$. It aims at accounting for the quantity of times that a node is visited and the quantity of times that a victory was obtained passing through this selfsame node.

2.3 Transposition Table

A transposition is the new occurrence of a state previously processed during the execution of the search algorithm. Transpositions in Go can occur due to situations in which, for example, pieces are captured and the board state returns to a previous configuration. The objective of a TT is to avoid that already processed data are reprocessed unnecessarily. As it is possible in a game of Go the capture of a single piece, as well as a group of pieces, after a capture move is executed, the board state may return to a configuration that is differentiated from the previous state, by one, two or many more pieces.

The TT is implemented as a hash table which is a data structure that associates keys to values. Each key represents a state on the Go game-board and is associated with a given piece of information such as, board evaluation and already explored tree depth. The representation of the board state in the form of hash key is carried out using a technique described by Zobrist (Zobrist 1970).

2.4 Hash Table and Hash Function

The use of the hash table allows for rapid data access from the TT, with average complexity per operation of *O(1)* and in the worst case *O(N)* (*N* is the size of the TT) (Szwarcfiter and Markenzon 1994), which may occur when it is not possible to conclude the data insertion operation, as all attempts collide, albeit all the positions are already occupied.

A good hash function should be easily computable and should produce a low number of collisions. One of the hash operations functions in the following manner, divide the hash key (x) by the size of the table (N) and the rest of the division (h) is used as an address for TT, as shown in equation 2:

$$h = x \, mod \, N. \quad (2)$$

2.5 Zobrist's Method

The method described by Zobrist (Zobrist 1970) is a technique of creating hash keys that use the XOR operator (or exclusive). The application of the XOR operator results in true if, and only if, only one of the operands is also true. Firstly, attribute a number pseudo-aleatory for each possible occupation for each position on the board. For example, a white piece in position [0,0] possesses a value, where a black piece in the same position possesses a different value.

The hash key created to represent each board state is the result of the XOR operation realized among all the elements associated with the board positions that contain pieces (Zobrist 1970).

3 Related Works

This section resumes some successful agents for Go (including Fuego and Gnu Go, which were involved in evaluative tournaments against BTT-Go).

The agent Fuego has its open source code available (Enzenberger et al. 2010), and it is, currently, among the best automatic player programs for Go. It operates in a strongly supervised way, since its search algorithm is based on MCTS simulations in which the construction of the tree is guided by *opening book* and *prior knowledge*. The *play-out* is based on hierarchical heuristic rules provided by expert human *play-out policy*. It is interesting to point out that, differently to BTT-Go, Fuego does not consider the archives of nodes' evaluation that are obtained throughout the searches performed in a game.

The Migos program (Mini Go Solver) is a player with supervised learning that uses alpha-beta search algorithm and search refining resources, such as symmetry research and TT. The values of the nodes that have already been evaluated and whose data is stored in the TT is used to release the alpha-beta algorithm of the burden of having it re-evaluated (van der Werf, Van Den Herik, and Uiterwijk 2003). For comparative purposes, the agent BTT-Go presented here, in addition to using the data from TT to reduce the number of evaluations, also uses it to reduce the number of move simulations in the *play-out* phase. The tests involving Migos presented in (van der Werf, Van Den Herik, and Uiterwijk 2003) intend on evaluating the performance of the agent only with respect to the reduction in the evaluation rate with the use of the TT (superior to 90%). No results involving tournaments against other agents were presented.

The MoGo player (Gelly et al. 2006) is based on MCTS simulations (not constructed from Fuego). Different to BTT-Go and Fuego, MoGo does not use a policy that balances quality and simulation frequency for insertion of nodes in the search tree (see equation 1). Instead, MoGo creates a group which contains board positions, where each group is filled in accordance with specialized knowledge from the game Go. Such a group is used only at the *tree construction* phase, in such a way that the search is restricted to moves that belong to this group (in the *play-out* phase this restriction is not applied). The definition of the moves in the *play-out* phase is orientated by standards established for sub-regions with board dimension 3x3, which characterizes a heuristic in the *play-out* phase. In tournaments against the agent GNU Go 3.6 (Bump 2003) in 13x13 game-boards, MoGo obtained a winning rate of 56%.

The Crazy Stone is a Go player program which makes use of MCTS (also not based on Fuego). Different to BTT-Go and Fuego, it does not use *prior knowledge* at the *tree construction* phase in MCTS. Instead, it uses a Bayesian technique, which defines the tree *selection* route based on evaluations Bradley-Terry (BT) (Coulom 2007a). The BT evaluator module operates together with a TT (Enzenberger and Müller 2010) which serves as a repository for previously evaluated nodes. The states evaluated by BT module are represented by features, which are mathematical functions that express relevant information concerning the board (Coulom 2007a). In (Coulom 2007a) the agent is evaluated through comparisons between the best move indicated by the Bayesian method and the move, which in the same situation would be indicated by human masters (available in database). In this test, Crazy Stone compared to other related works, showed good prediction of moves from the mid game stage. In other tests, Crazy Stone played against the agent GNU Go 3.6, obtaining a winning rate of 90.6% and 57.1%, respectively, in 9x9 and 19x19 game-boards.

The player Gnu Go (Bump 2003) uses the MCTS algorithm as in MoGo, that is, the player uses supervised learning through board standards (Bump 2003). In matches played on boards 19x19 against Fuego, the Gnu Go obtained a 33.3% victory rate playing with black pieces and 20% playing with white pieces.

4 BTT-Go

The system BTT-Go is inspired on the open source agent for Go Fuego. It has as its main purpose to reduce the supervised character of the latter by means of four strategies: first, to replace the heuristic values of the nodes used by Fuego in the *tree construction* phase (*prior knowledge*) with the values of the TT that were obtained in previous evaluations; second, to interrupt the *play-out* simulations whenever their paths find a node whose value is stored on the TT; next, to use the information of the TT to update the values of the search tree nodes during the *back-propagation* phase; and, finally, to use the new values obtained in the *back-propagation* to update or to add new information into the TT (which allows the TT to accumulate the history of each node evaluation). In order to make a trade-off between the loss of performance (due to the reduction in the supervision resources) and the gain of autonomy (with the insertion of the TT) BTT-Go will use the TT only from the point where more than 50% of the positions of the game-board is filled in with a piece. It is interesting to point out that the next section presents the architecture of BTT-Go and the interactions among its modules during the search process.

4.1 Architecture of BTT-Go

As presented before, BTT-Go tries to reduce the heuristic supervision of Fuego by using the information related to the history of the nodes' evaluation retrieved from the TT. Figure 1 illustrates the architecture of this agent. In order to choose a move, the current game-board *sc* is presented to the search algorithm, #1 (link 1 of the figure). This algorithm then constructs the MCTS tree by recursively executing the *n* possible episodes within the established interval of time. For each node *s* involved in these episodes (either during the construction of the tree or in the *play-out* phase), the algorithm presents *s* to the hash-key module, #2. The latter calculates the hash-key value corresponding to *s* and checks whether this state belongs or not to the TT, #3. If it does not (which means that it has not been evaluated yet), in link #4, like Fuego, the following proceeding is executed: if *s* is involved in the tree construction phase, the search algorithm continues normally with the construction of the tree considering that *s* has the value indicated by the *prior knowledge*. Otherwise, if *s* is a *play-out* node, the simulation of the current episode goes on normally according to the *play-out policy*. As soon as the *back-propagation* phase is concluded, the new values of all nodes involved in the current episode that already belong to the TT (including *selection* nodes, *expansion* nodes and *play-out* nodes), are used to update the TT. On the other hand, those which still do not belong to the TT are introduced into it, increasing its information data. Then, as BTT-Go also inserts the *play-out* nodes into the TT, it does not waste the history of their evaluations obtained throughout the execution of the episodes (different to Fuego), which represents one of the contributions for the use of the TT. In link #5, which corresponds to the situation where *s* belongs to the TT, the following procedure is executed: if *s* is a *selection* node or an *expansion* node, the tree construction goes on taking into account its TT value. This TT value keeps information about all the history related to the evaluation of *s* that has been obtained throughout the game. In a distinct way, Fuego, in the same situation, uses the following values: if *s* is an *expansion* node, it uses its heuristic *prior knowledge* value; otherwise, if *s* is a *selection* node, Fuego uses its updated value that only keeps information of the current search process. Then, by using the TT values (whenever they are available) instead of using the *prior knowledge* heuristics, BTT-Go reduces the supervised character of Fuego. On the other hand, if in link #5 *s* is a *play-out* node, the BTT-Go search algorithm interrupts the simulation of the current episode. As game simulations correspond to a kind of supervision (since the results of the simulated games will guide the agent in the choice of the appropriate move to be executed), this reduction in the rate of simulations represents a supervision reduction. Uniquely in this case, the updating of the values of all nodes during the *back-propagation* will be calculated based, exclusively, on the following parameters: their current values and their corresponding *play-out* statistics (see section 2.2). This is due to the fact that, as the *play-out* has been interrupted, there is no new information concerning the rating of victories. Obviously, these new values are also used to alter the TT, by updating its values or by increasing its information data. Finally, as soon as the *n* episodes are concluded, the search algorithm points out the best move to be executed from *sc*, #6. The new game-board obtained with this execution becomes the new current game-board, #7, from which *n* other new episodes will be executed in order to choose the next move. As aforementioned, the TT values are continously updated during the backpropagation such that they keep the history of all the node evaluations that were performed throughout a game. Therefore, they are appropriate to substitute the *prior-knowledge* heuristics without compromising the accuracy of the move evaluations. That is why the TT is a reliable tool for reducing the supervison in the player agent. The next section resumes the generation of the hash keys and the structure of the TT in BTT-Go.

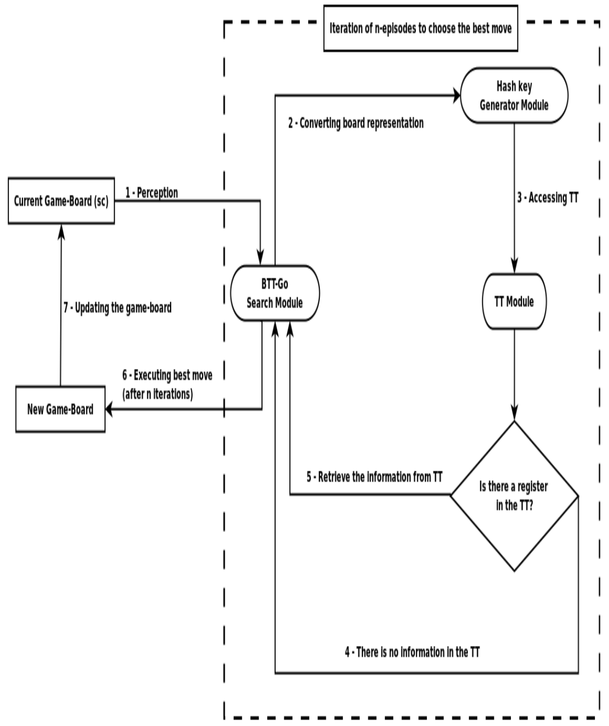
Iteration of n-episodes to choose the best move
Hash key Generator Module
2 - Converting board representation
Current Game-Board (sc)
1 - Perception
3 - Accessing TT
BTT-Go Search Module
TT Module
7 - Updating the game-board
6 - Executing best move (after n iterations)
New Game-Board
5 - Retrieve the information from TT
Is there a register in the TT?
4 - There is no information in the TT

Figure 1: Architecture of BTT-Go.

4.2 Hash Keys and TT

The previous section presented the general architecture of BTT-Go and described the interactions among them during the search process. This section adds some relevant details about the hash key generation and the TT module.

Hash key Generator Module In BTT-Go, the address of every state *s* in the TT is referenced by means of two hash keys (with 64 and 32 bits, respectively, considering the first and the second hash keys). Both keys are generated by the Hash key Generator Module as it follows: initially, an arbitrary and distinct integer number is chosen to represent every possibility of occupation (that is, with black or white pieces) in each position of *s*. Next, the Zobrist method generates each hash key of *s* by executing XOR operations among the integers that correspond to that key. The 32 bits hash key is used to solve collision problems (Szwarcfiter and Markenzon 1994) that may occur when accessing the TT.

TT Module In the BTT-Go the TT Module uses the TableEntry class to define the attributes which make up a data entry in the TT, as shown in the following:

```
class TableEntry {
        long long int keyHash1;
        long int keyHash2;
        int treeDepth;
        double evaluation;
}
```

The attributes keyHash1 and keyHash2 represent the 64 and the 32 bits hash keys, respectively (which are used to calculate the address in which a given state *s* is stored on the TT). The treeDepth attribute indicates the depth of the search in which *s* was explored. The evaluation attribute is supplied to the search algorithm indicating the occurrence of a transposition.

For each game-board *s* inserted into the TT, three other additional registers containing symmetric configurations of *s* are also inserted into it. These configurations are obtained by applying to *s* rotations of 90°, 180° e 270° degrees. This strategy enriches still further the knowledge stored in the TT.

The existence of game cycles, which consist of *Ko* occurrence (consecutive repetition of two moves in cycle) is avoided, as this impedes reaching an end game state.

In relation to the use of the TT, it is worth highlighting that the values of the search algorithm recuperates from the TT during the *tree construction* phase correspond to the portion $Q^{current}_{(s,a)}$ from equation 1. This represents a contribution in relation to Fuego, once that the latter uses, for such a portion, the *prior knowledge* heuristics for the *expansion* phase, or the current value (which does not consider the evaluation archive), in the *selection* phase. On the other hand, the portion $Simul_{(s,a)}$ from the same equation is calculated analogously to Fuego. When it comes to the portion $Vict_{(s,a)}$ of the referred equation, relative to the *play-out* phase, if the node *s'* visited by BTT-Go (child of *s* obtained by the execution of action *a* from *s*) belongs to TT, the *play-out* phase will be interrupted and such a portion will not be considered (as described in the previous section). On the other hand, if the visited node does not belong to TT, it will be calculated such as in Fuego.

5 Experiments and Results

The quantitative impact of the supervision reduction in BTT-Go is evaluated here through the reduction in the number of play-out simulations (and consequently, in the search runtime) observed in games against Fuego. The average number of *play-out* simulations executed by BTT-Go in 20 moves was 30% inferior to that of Fuego (obviously due to the use of the TT). Consequentely, the BTT-Go search runtime is also 30% less than that of Fuego. The qualitative impact of the supervision reduction in BTT-Go is estimated in this work by observing its winning rate in tournaments against Gnu Go and Fuego. For the purpose of analyzing the rate of victories, three test scenarios were performed. In all of them, each agent plays half of the games with black pieces and the remaining ones with white pieces, since the black player has the significant advantage of executing the first move. In real tournaments, in order to attenuate a little this advantage, normally, the white player receives a determined bonus (called *komi*). Particularly here, this bonus corresponds to 6.5 points (that tries to compensate the cited advantage). Further, with the objective of focusing only in the effect of using the information retrieved from the TT instead of using the *prior knowledge* and the *play-out* heuristics (whenever it is possible), in the evaluative tournaments executed here no agent counts on *opening book* heuristics. All scenarios evaluate the accomplishment of the agents in 9x9, 13x13 and 19x19 game-boards. The configuration of the computer used in the experiments is: *Intel Core 2 Quad* 2.4 GHz with 8 GB of RAM.

By this way, in scenario 1 BTT-Go faces Gnu Go in tournaments composed of 30 games for each game-board dimension. Table 2 shows the winning rate of BTT-Go against its adversary (as black player, 100%, 100% and 86.7%, respectively, for 9x9, 13x13 and 19x19 game-boards; as white player, 100%, 93.3% and 60%, respectively, for 9x9, 13x13 and 19x19 game-boards) and confirms that its performance is significantly superior in all situations.

Table 2: Scenario 1 – Winning Rate of BTT-Go X Gnu Go.

Board Size	Black	White
9x9	100% (± 0)	100% (± 0)
13x13	100% (± 0)	93.3% (± 6.4)
19x19	86.7% (± 8.8)	60% (± 12.6)

Scenario 2 involves BTT-Go and Fuego in tournaments of 100 games for each game-board dimension. Table 3 presents the winning rate of BTT-Go against its adversary (as black player, 44%, 42% and 42%, respectively, for 9x9, 13x13 and 19x19 game-boards; as white player, 34.7%, 34% and 26%, respectively, for 9x9, 13x13 and 19x19 game-boards). The results show that BTT-Go, in spite of counting on much less supervision than Fuego, it is still very competitive, even playing on 19x19 game-boards, especially when playing with black pieces. Obviously, as was predicted, the accomplishment of the agent is compromised when it plays with white pieces, mainly in 19x19 game-boards. But considering the disadvantage of BTT-Go compared to Fuego in terms of supervision level and the results obtained by other adversaries (as Gnu Go - see section 3) in the same difficult situation of facing Fuego in 19x19 game-boards, it is reasonable to consider that BTT-Go was not completely "butchered" playing against one of the best current agents of Go.

Finally, the main objective of scenario 3 is to check whether the information retrieved from the TT in BTT-Go is, in fact, a good alternative compared to the knowledge expressed in the *play-out* heuristic rules that BTT-Go inherited from Fuego. To cope with this goal, scenario 3 uses a modified version of BTT-Go. In the *play-out* phases of this

Table 3: Scenario 2 – Winning Rate of BTT-Go X Fuego.

Board size	Black	White
9x9	44% (± 7)	34.7% (± 6.8)
13x13	42% (± 7)	34% (± 6.7)
19x19	42% (± 7)	26% (± 6.2)

altered version, each time that a move M indicated by the TT of BTT-Go is not the same as the one which would have been pointed out by the *play-out* rules, the value of M retrieved from the TT is reduced by 30%. This corresponds to a "punition" applied to BTT-Go for disagreeing with the heuristics rules. The results presented in Table 4 show that the TT values really present an accuracy that makes them reliable enough to replace the *play-out* heuristics. In fact, the performance of the altered version of BTT- Go clearly decreases when it is compared to the original version of the agent (as black player, 56%, 30% and 20%, respectively, for 9x9, 13x13 and 19x19 game-boards; as white player, 26%, 30% and 26%, respectively, for 9x9, 13x13 and 19x19 game-boards). It is interesting to point out that in this third scenario, playing in 9x9 game-board, BTT-Go, different to what happened in the tournaments in 13x13 and 19x19 game-boards, improved significantly its performance. It can be explained by the fact that in a dispute in small game-boards, such as 9x9, there occur fewer simulations of moves than in a dispute on larger game-boards, which reduces a little the reliability of TT information.

Table 4: Scenario 3 – Winning Rate of Altered-BTT-Go X Fuego.

Board size	Black	White
9x9	56% (± 7)	26% (± 6.2)
13x13	30% (± 6.5)	30% (± 6.5)
19x19	20% (± 5.7)	26% (± 6.2)

6 Conclusion and Future Works

This article presented the creation of an agent for Go, BTT-Go, which has as its objective to reduce the supervision of the very successful automatic player Fuego with the incorporation of a TT, which operated with MCTS. Such a TT serves as tools used to store and update the state values throughout a match, as they reappear by transposition. Therefore, the values of the nodes stored in TT accumulates the whole history of the evaluations in the context of the game, by means of successive updates (and not only the informative content of the current search or heuristic content of the simulations that are used by Fuego). The MCTS algorithm in Fuego has its good performance tied up in the quantity of simulations of realized games (the greater the perceptiveness of the state evaluations during the search) and the use of *prior knowledge* and *play-out* heuristics as auxiliary tools in the search for the best move. Following this direction, the BTT-Go agent will, always when possible, substitute the use of Fuego heuristics for information provided in TT, which provides a good perceptiveness in the node evaluation, even with the reduction in supervising recourses of Fuego. As presented in the results section, the good performance associated with BTT-Go in tournaments against Fuego and Gnu Go, including those on 19x19 boards, proves that the technique used herein permitted a greater autonomy to the agent in terms of supervision, without significantly compromising its performance. As future works the authors will continue to work on adapting the agent search system to a high performance environment, which should improve greatly its operational capacity, mainly in the *play-out* phase.

References

Brügmann, B. 1993. Monte carlo go.

Bump, D. 2003. Gnugo home page.

Campos, P., and Langlois, T. 2003. Abalearn: Efficient self-play learning of the game abalone. In *INESC-ID, Neural Networks and Signal Processing Group*.

Chaslot, G.; Saito, J.-T.; Bouzy, B.; Uiterwijk, J.; and Van Den Herik, H. J. 2006. Monte-carlo strategies for computer go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, 83–91.

Childs, B. E.; Brodeur, J. H.; and Kocsis, L. 2008. Transpositions and move groups in monte carlo tree search. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, 389–395. IEEE.

Coulom, R. 2007a. Computing elo ratings of move patterns in the game of go. In *Computer games workshop*.

Coulom, R. 2007b. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*. Springer. 72–83.

Enzenberger, M., and Müller, M. 2010. A lock-free multithreaded monte-carlo tree search algorithm. In *Advances in Computer Games*. Springer. 14–20.

Enzenberger, M.; Muller, M.; Arneson, B.; and Segal, R. 2010. Fuego – an 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.; Teytaud, O.; et al. 2006. Modification of uct with patterns in monte-carlo go.

Junior, E. V., and Julia, R. M. S. 2013. Btt-go: Um agente para go baseado em simulacoes, tabela de transposicao e modelo bradley-terry. Not published, oral presentation in a local event held at Federal University of Uberlandia.

Szwarcfiter, J. L., and Markenzon, L. 1994. *Estruturas de Dados e seus Algoritmos*, volume 2. Livros Técnicos e Científicos.

van der Werf, E. C.; Van Den Herik, H. J.; and Uiterwijk, J. W. 2003. Solving go on small boards. *ICGA Journal* 26(2):92–107.

Zobrist, A. L. 1970. A new hashing method with application for game playing. *ICCA journal* 13(2):69–73.