TELKOMNIKA, Vol. 11, No. 2, February 2013, pp. 667~673
e-ISSN: 2087-278X

Application of MCTS in Tsumego of Computer Go

Fang Wang*, Ying Peng
Institution College of Computer Science and Technology Southwest University for Nationalities, Chengdu, China. addres, 0086-18981790887/0086-028-85522523
*Corresponding author, e-mail:livenski@163.com

Abstract

The Tsumego problem in Go was a basic and essential problem to be overcome in implementing a computer Go program. This paper proposed a reality of Monte-Carlo tree search in Tsumego of computer Go which using Monte-Carlo evaluation as an alternative for a positional evaluation function. The advantage of this technique was that it requires few domain knowledge or expert input.

Keywords*: Tsumego, Monte-Carlo tree search, the law of large numbers, computer Go*

1. Introduction

Tsumego problems have been found in Chinese books dating back to around the 13th century. They are presumably composed and collected from actual games much earlier. They range from situations that occur quite commonly, which every strong player ought to be familiar with, to deliberately difficult puzzles. Some books of the latter type are still used for professional training.

Several conventions are used in the problems. The objective is to kill a group or prevent it from being killed. Problems do not specify how many plays are in the solution (as would be usual in a chess problem), because the goal of the problem is rarely to capture stones; as soon as the correct first move is played, the threatened group can be considered alive (or dead). Solution diagrams will either show the most tenacious resistance that the opponent can offer, or lines that require interesting or tricky tactics. If only part of the board is shown, as is usually the case, the rest of the board can be assumed to be empty. The modern convention is that well composed problems do not allow the threatened group to escape into empty areas of the board (this is one way in which such problems differ from real games), although escape and recapture was a theme in classical problems.

The Monte Carlo method is a technique for analyzing phenomena by means of computer algorithms that employ, in an essential way, the generation of random numbers. The Monte Carlo method was given its name by Stanislaw Ulam and John von Neumann, who invented the method to solve neutron diffusion problems at Los Alamos in the mid1940s.

Monte Carlo methods are widely used in mathematics, science, industry, commerce, and entertainment. They are at the heart of algorithms used to make predictions about stochastic processes, that is, phenomena having some random component. This includes the motion of microscopic particles in an environment, the generation and movement of data packets through networks, the arrival and servicing of vessels at a busy port, and hundreds of other processes about which people need answers. Random numbers are used directly in the transmission and security of data over the airwaves or along the Internet. A radio transmitter and receiver could switch transmission frequencies from moment to moment, seemingly at random, but nevertheless in synchrony with each other. The Internet data could be credit-card information for a consumer purchase, or a stock or banking transaction secured by the clever application of random numbers. And randomness is an essential ingredient in games of all sorts, computer or otherwise, to make for unexpected action and keen interest [1].

The simplest method for solving the Tsumego problem is to explore all the possible moves until all terminal moves are encountered, and then to determine the best move sequence for both players by a heuristic evaluation function. This approach requires enormous computing time, and thus we cannot use this kind of exhaustive searching in reality. So Monte-Carlo Tree Search (MCTS) is one key to solve the Tsumego problem in computer Go.

Received August 29, 2012; Revised December 23, 2012; Accepted January 11, 2013

2. Research Method

2.1. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a best-first search method that does not require a positional evaluation function. It is based on a randomized exploration of the search space. Using the results of previous explorations, the algorithm gradually builds up a game tree in memory, and successively becomes better at accurately estimating the values of the most-promising moves.

In Tsumego of computer games, one possible Evaluation Function (EF) is the Monte-Carlo (MC) method. Given a board position B, its aim is to compute a value V(B) for this position. Starting from the position B, MC plays a certain number of simulated games. A simulated game is a succession of moves (called simulated moves), played until the end of the game is reached. The MC evaluation V(B) is then deduced from the results of all the simulated games. In the simplest version, simulated moves are random moves, and V(B) is the average of the outcomes of the simulated games.

MCTS is mainly consists of four main steps: selection, expansion, simulation, and backpropagation. Each step has a strategy associated that implements a specific policy. In the selection step, the tree is traversed from the root node once we reach a node E and select a child of its which is not part of the tree. In the expansion step, that child is added to the tree. In the simulation step, moves are played in self-play until the end of the Tsumego and return it's output. In the backpropagation step, the output is propagated backwards through the previous traversed nodes and the move played by the program is the child of the root with the highest visit count.

2.2. Monte-Carlo Tree Search in Tsumego of Computer Go

MCTS is applicable if three conditions are satisfied: (1) the target is locked, (2) complete information, and (3) simulations terminate relatively fast. Tsumego meets all the conditions perfectly. Tsumego's Target is clear-cut as present in section 1. Go has it's clear rules. Tsumego is a small part of game, it's length is limited of course. So MCTS is one of best solution to Tsumego of computer Go.

The selection operated in the following way. From the initial local position, a selection strategy was applied recursively until a position was reached that was not a part of the sequence yet. At each move, the player could select one of few points of the position, which leaded to death or life. The more number of simulations was the more accurate of selection strategies for Tsumgego.

In the expansion procedure, for the next move was so limited to full game, all children of a node were added to the tree when a certain number of simulations had been made through this node. Of course, one may forbid any node expansion to save memory before certain number of simulations had been made through this node.

In the simulation procedure, progam selected moves itself until the game was over. The simulation strategy was the key to good performance gain in an MCTS program. The strategy should not become too stochastic nor too deterministic. At first, an urgency value Ui, was computed for each move i. At the second, taking the ungencies into account a move is randomly drawn. The probability of each move to be selected was calculated by Formulate (1).

$$p_{i=} \frac{Uj}{\sum_{k \in N} U_k} \quad (1)$$

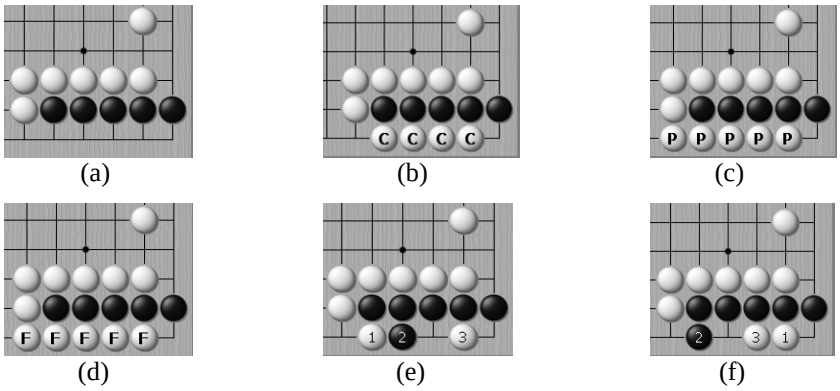
where N is the set of all possible moves for a Tsumego problem.

In the procedure of backpropagation, program propagated the result of a simulated game k backward from leaf node L to the nodes it had to traverse to reach this leaf node. The result was counted positively (Rk = +1) if the game was won, and negatively (Rk = −1) if the game was lost. Draws lead to a result Rk = 0. A backpropagation strategy was applied to compute the value vL of a node.

The most popular and most effective strategy was Average, which toke the plain average of the results of all simulated Tsumego made through this node ,i.e., $v_L = (\sum_k R_k) / n_L$.

2.3. Move Selection in Tsumego of Computer Go

Human players selected their own vital point in a few seconds by Go sense, which was learnt from experience during they had played games. And then they deeply examined the best sequence of moves for both players based on the possible vital point. For Tsumego, eyes shape analysis was the only one method to extract a set of first moves.



(a) (b) (c)

(d) (e) (f)

Figure 1. Illustration of A Life-and-death Problem and A Set of First Moves. (a) A Life-and-death Problem. (b) Candidate First Moves. (c) Possible Vital Moves. (d) A Set of First Moves For Game Tree Searching. (e) and (f) The Sorrect sequence of Moves to Kill Black's Group.

The eye shape analyzer generated a set of possible vital moves. Figure 1(c) show the three points which were the set of possible vital moves generated by the heuristic influence function. The combined set of candidate first moves and possible vital moves became a set of first moves in Figure 1(d), for solving the life-and-death problem in Figure 1(a). Figure 1(d) illustrated the set of first moves to create the nodes in the first level of the game tree. Both Figure 1(e) and Figure 1(f) were the correct sequence of moves to kill black's group.

The eye shape analyzer generated a set of possible vital moves. Figure 1(c) show the three points which were the set of possible vital moves generated by the heuristic influence function. The combined set of candidate first moves and possible vital moves became a set of first moves in Figure 1(d), for solving the life-and-death problem in Figure 1(a). Figure 1(d) illustrated the set of first moves to create the nodes in the first level of the game tree. Both Figure 1(e) and Figure 1(f) were the correct sequence of moves to kill black's group.

Each Tsumego i that was played had a result Ri. Let R be the random variable which took the values Ri. The Ri values were bounded, so R had an average value μ=E(Ri) and a standard deviation σ=D(Ri). In the Tsumego, Ri often belongs to [-100,100] and σ is usually lower than 50.

Let $\{R_1, R_{2,} \cdots, R_n\}$ be a random sample of size n—that was, a sequence of independent and identically distributed random variables drawn from distributions of expected values given by μ and finite variances given by σ2. We were interested in the sample average $S_n = (R_1 + \cdots + R_n)/n$ of these random variables. By the law of large numbers, the sample averages converged in probability and almost surely to the expected value μ as n tends to infinity. The classical central limit theorem describes the size and the distributional form of the stochastic fluctuations around the deterministic number μ during this convergence. More precisely, it states that as n gets larger, the distribution of the difference between the sample average Sn and its limit μ, when blown up by the factor , approximates the normal distribution with mean 0 and variance σ2. For fixed large n one can also say that the distribution of Sn is close to the normal distribution with mean μ and variance $\frac{1}{n}\sigma^2$. The usefulness of the theorem is that the distribution of $\sqrt{n}$ approaches normality regardless of the shape of the distribution of the individual Ri's.

Thus a confidence interval for Sn can be deduced. Let Sn be the value of the average that we would expect after an infinite number of stochastic games. There was 66% of confidence that Sn was within the interval $[S_n - \sigma/\sqrt{n}, S_n + \sigma/\sqrt{n}]$, 95% confidence that Sn was within the interval, etc. Move Selection in Tsumego of computer Go was based on the results provided by this theorem.

2.4. Move Selection in Tsumego of Computer Go

Monte Carlo Tree Search is an optimal limited search methods which combining offline learning knowledge and online search. In the procedure of MCTS, we make online searching by method of UCT. We modify the prune method of UCT algorithm which is the basis of online searching to update it to MCTS algorithm.

Pruning condition's obtain are only dependent on the property of which the UCB algorithm to solve multi-armed bandits problem. Suppose that there is a K arm multi-armed bandits model, a_i represents the i-th arm， v_i represents the times that the i-th arm accessed currently, we represent times of the i-th arm won currently. $v = \sum_i v_i$ represents the sum of times of all the arm accessed by now, $w = \sum_i w_i$ represents the sum of times of all the arm winning, there $i \in [1, k]$.We know $v_i \geqslant w_i$ and $v \geqslant w$ because the winning times are always limited by the times of accessed. Now we use $\overline{v_i}$ represents the times which i-th arm is to be accessed and $\overline{w_i}$ represents the times of that i-th arm will win. $\overline{v_i} = \sum_i \overline{v_i}$ represents the total sum of times of each arm to be accessed. $\overline{w_i} = \sum_i \overline{w_i}$ represents the total sum of times of win, which is going to access. Same as above, Obviously, $\overline{v_{\_1}} \geq \overline{w_{\_1}}$ and $\overline{v_i} \geq \overline{w_i}$. $V = v + \overline{v}$ reprents the expected total access times.

The pruning conditions can be expressed as: If $\exists j$ make $v_j > \frac{V}{2}$, then a_i can be pruned when $j \in \{1, ...K\} \cap i \neq j$.

Obviously, if

$$v_j > \frac{V}{2} \qquad j \in \{1, ...K\} \cap i \neq j \tag{2}$$

Then $$\sum_{i \neq j} v_i = V - v_j < \frac{V}{2} \tag{3}$$

therefore $v_i < V/2 < v_j$ (4)

We will always choose the next point which is accessed to the most frequently when we determine which point will become to final point with UCT algorithm.It ensures the final decision results which made according to the access times consistent with the results of original UCT because the node which is most frequently accessed will not meet the pruning conditions when using the pruning conditions.

Perhaps there are nodes exceed half of the expected when the total access times is not reached that we expected if there exist the point that significantly better than the other nodes. Therefore, time is saved and simulation is finished in advance by add pruning conditon.

We do not using all kinds of pruning conditions all the time for the following two points: 1) For each node has been joined, it is necessary take a large amount of time to calculate the each related parameters which is needed by each pruning condition if we judge it whether satisfies Pruning conditions every moment when access the node. 2) All kinds of state of the

node are unsTable when it is joined to game tree. So, we will divide each round of simulated time to N segment (1 second), and make a pruning to all the current nodes at the end of each time segment.

3. Results and Analysis

3.1. Less State Space

To Game tree, it was consider that the number of nodes is the number of the state space for branch of Game tree constantly expanding-down. However, huge state space is a key factor which limites the accuracy of game tree search. The number of state space discreases after Combing game tree with idea of diagram for it only considers amount of true state space. And it makes evaluation more accracy to state space with sharing data.

The following experimental data is the contrast between the game using replacement Table and those not using replacement Table. Table 1 canlenders average mumber per second of different expand time.

Table 1. State number of game tree and vertex state number of game diagrame at diffrent time

Time	Number of state without using replacement Table	Number of state with using replacement Table
15	1	1
13	2	1.2
11	3	2.1
9	5	3.2
7	8	5.3
5	11	8.7
3	16	13.6
1	21	21

We can see from the Figure, with the increase in access time, the reduction in state space become apparent relatively. That is, the more game tree branch expansing, the more obvious effect of replacement Table.Due to the sharing of data, we can make more quick for exploring.

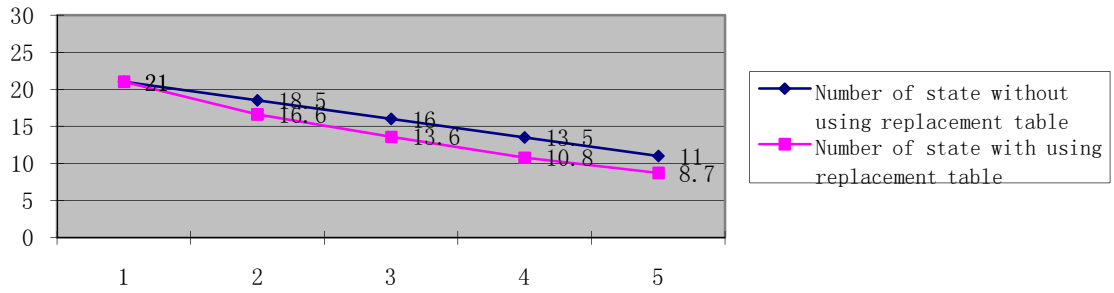


Figure 2. Line Chart of State Number Of Game Tree and Vertex State Number of Game Diagrame At Diffrent Time

3.2. Superior Ablebility to Solve Tsumego Problem

We chose XIU XING TSUMEGO PROBLEM MASTERPIECE SETS which is written by famous professional player Hideyuki Fujisawa. It contains four difficulty Tsumego problem (primary, intermediate, high level and high Duan). Especially the Tsumgo problem of high Duan difficulty, it is complex even professional player need long time to give right answer. Table 2 shows the answer circumstances of these four difficulty Tsumego problem. To primary, intermediate and high level difficulty Tsumego problem, we set 5 minutes as it's time limit. Tsumego problem solving is failure if solving time exceeds 10 minutes. And to high duan difficulty Tsumego problem, 60 minutes is it's time limit.

Table 2. Answer Circumstances of Four Difficulty Tsumego Problem

difficulty	Number of Tsumego problem	Number of Right answer	Correct Rate	Longest right answer time	Shortest right answer time	Average right answer time
Primary	40	38	0.95	4.367	0.001	0.612
Intermediate	39	35	0.9	5.332	0.001	0.287
High level	61	54	0.89	12.533	0.003	0.934
High Duan	23	18	0.78	1898.9	0.006	145.6

Table 3 shows five Tsumego problem of high Duan difficultythe which spend longest time to answer it. Especially the 163th Tsumego problem, Hideyuki Fujisawa had expressed that even the professional player could not resolve it without a few hours thinking.

Table 3. Answer Circumstances of High Duan

Number of Tsumego problem	Empty point of Tsumego problem	Searching situation	Max seaching depth	Answer time
162	25	836218537	75	1898.8
154	22	124920345	74	342.6
163	21	71483243	55	170.3
161	20	4020458	99	95.6
149	22	4937646	53	13.8

4. Conclusion

Monte-Carlo method generated a list of potential moves, and for each move playing out hundreds of games at random on the resulting board of Tsumego. It was a best-first search method that does not require a positional evaluation function in contrast to search. MCTS was based on a randomized exploration of the search space. Using the results of previous explorations, MCTS gradually build a game tree in memory, and successively becomes better at accurately estimating the values of the most promising moves. The advantage of this technique was that it requires very little domain knowledge or expert input, the trade-off being increased memory and processor requirements. Dislike the whole game, the moves used for evaluation are generated at random and it is possible that a move which would be excellent except for one specific opponent response would be mistakenly evaluated as a good move. Monte-Carlo Method in Tsumego cared neither overall strategic sense nor tactically, so we don't consider domain knowledge.

References

[1] Ronald W Shonkwiler, Franklin Mendivil. *Explorations in Monte Carlo Methods.* Springer. 2009
[2] Abramson B. Expected-Outcome: A General Model of Static Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1990; 12(2): 182–193.
[3] Allis LV, Meulen M, Van Der, Herik HJ. Proof-Number Search. *Artificial Intelligence*. 1994; 66(1): 91–123.
[4] Auer P, Cesa-Bianchi, Fischer P. Finite-Time Analysis of the Multi-Armed Bandit Problem. *Machine Learning*. 2002; 47(2–3), 235–256.
[5] Barto AG, Bradtke SJ, Singh SP. Learning to Act using Real-Time Dynamic Programming. *Artificial Intelligence*. 1995; 72(1–2), 81–138.
[6] Baxter J, Tridgell A, Weaver L. Experiments in Parameter Learning Using Temporal Differences. *ICCA Journal*. 1998; 21(2), 84–99.
[7] Beal DF, Smith MC. Temporal Difference Learning for Heuristic Search and Game Playing. *Information Sciences*. 2000; 122(1): 3–21.
[8] Berger F. BGBlitz Wins Backgammon Tournament. *ICGA Journal*. 2007; 30(2): 114.
[9] Berliner HJ. The B*-Tree Search Algorithm: A Best-First Proof Procedure. *Artificial Intelligence.* 1979; 12: 123–40.
[10] Boer PT de, Kroese DP, Mannor S, Rubinstein RY. A Tutorial on the CrossEntropy Method. *Annals of Operations Research*. 2005; 134(1): 19–67.
[11] Bouzy B. Mathematical Morphology Applied to Computer Go. *International Journal of Pattern Recognition and Artificial Intelligence.* 2003; 17(2): 257–268.

[12] Bouzy B. Associating Domain-Dependent Knowledge and Monte Carlo Approaches within a Go Program. *Information Sciences, Heuristic Search and Computer Game Playing IV*. 2005; 175(4): 247–257.
[13] Bouzy B, Cazenave T. Computer Go: An AI Oriented Survey. *Artificial Intelligence.* 2001; 132(1): 39–103.
[14] Buro M. Toward Opening Book Learning. *ICCA Journal*. 1999; 22(2): 98–102.
[15] Campbell M, Hoane AJ, Hsu FH. Deep Blue. Artificial Intelligence. 2002; 134(1–2): 57–83.
[16] Chaslot GM J-B, Winands MHM, Herik HJ van den. Parallel Monte Carlo Tree Search. Computers and Games. 2008; 5131 of LNCS: 60–71.
[17] Chaslot GM J-B, Winands MHM, Szita I, Herik HJ van den. CrossEntropy for Monte-Carlo Tree Search. *ICGA Journal*. 2008; 31(3): 145–156.
[18] Chaslot GM J-B, Winands MHM, Herik HJ van den, Uiterwijk JWHM, Bouzy B. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*. 2008; 4(3): 343–357.
[19] Chen Z. Semi-Empirical Quantitative Theory of Go Part I: Estimation of the Influence of a Wall. *ICGA Journal.* 2002; 25(4): 211–218.
[20] Coulom R. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. Computers and Games. 2006; 4630 of LNCS: 72–83.
[21] Gelly S, Wang Y. MoGoWins 19×19 Go Tournament. *ICGA Journal*. 2007; 30(2): 111–112.
[22] Winands MHM, Herik HJ van den. Proof-Number Search and Its Variants. Oppositional Concepts in Computational Intelligence. 2008; 155: 91–118.
[23] Kloetzer J, Iida H, Bouzy B. Playing Amazons Endgames. *ICGA Journal*. 2009; 32(3): 140–148.
[24] Lee CS, Wang MH, Chaslot, GMJB, Hoock JB, Rimmel A, Teytaud O, Tsai SR, Hsu SC, Hong TP. The Computational Intelligence ofMoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*. 2009; 1(1): 73–89.
[25] Lishout F, Chaslot GMJB, Uiterwijk JWHM. *Monte-Carlo Tree Search in Backgammon*. Proceedings of the Computer Games Workshop. 2007; 175–184.
[26] Schadd MPD, Winands MHM, Herik HJ van den, Chaslot GMJB, Uiterwijk JWHM. Single-Player Monte-Carlo Tree Search. *Computers and Games*. 2008; 5131 of LNCS; 1–12.
[27] Schadd MPD, Winands MHM, Herik HJ van den, Chaslot GMJB, Uiterwijk JWHM, Bergsma MHJ. 2008; 4(3): 369–384.
[28] Audibert JY, Bubeck S. *Minimax Policies for Adversarial and Stochastic Bandits*. Proceedings of the 22nd Annual Conference on Learning Theory. 2009.
[29] Chaslot G, Bakkes S, Szita I, Spronck P. *Monte-Carlo Tree Search: A New Framework for Game AI*. Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference. 2008; 216–217.
[30] Gelly S, Silver D. *Combining Online and Offline Knowledge in UCT. ICML ' 07*: Proceedings of the 24th International Conference on Machine Learning 2008; 273–280.
[31] Gelly S, Hoock JB, Rimmel A, Teytaud O, Kalemkarian Y. *The Parallelization of Monte-Carlo Planning Parallelization of MC-Planning*. Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics, Intelligent Control Systems and Optimization. 2008; 244–249.