A.I for Games with High Branching Factor

Sabarinath Mohandas
Department of Computer Science and Engineering
College of Engineering Trivandrum
Thiruvananthapuram, India
sabarinathm@hotmail.com

M Abdul Nizar
Department of Computer Science and Engineering
College of Engineering Trivandrum
Thiruvananthapuram, India
nizar@cet.ac.in

***Abstract*—Monte Carlo tree search (MCTS) is a heuristic search method that is used to efficiently search decision trees. The method is particularly efficient in searching trees with a high branching factor. MCTS has a number of advantages over traditional tree search algorithms like simplicity, adaptability etc. This paper is a study of existing literature on different types of MCTS, specifically on using Genetic Algorithms with MCTS. It studies the advantages and disadvantages of this approach, and applies an enhanced variant to Gomoku, a board game with a high branching factor.**

***Index Terms*—Artificial intelligence (AI), game playing, Monte Carlo tree search (MCTS), genetic algorithms (GA), gomoku, Connect-5.**

I. Introduction

A decision tree is a commonly used decision support tool. It uses a tree-like graph, showing the current state, the decisions that can be taken, their possible outcomes and resource costs. The number of decisions that have to be evaluated to find the best one depends on the complexity of the problem at hand. For very complex problems the decision tree has a large number of possible actions. Most of these paths are low reward actions. The intuition of human mind is very efficient in such a scenario, wherein all possibilities are quickly evaluated, and low reward paths are promptly eliminated. This allows the human mind to concentrate on the few remaining high reward paths. Machine A.I, on the other hand lacks this intuition ability, forcing it to evaluate each and every path before deciding which one is the best.

An example of tic-tac-toe, the problem and its branches are shown below. It is a directed graph containing all possible moves from each position; the nodes are positions in a game and the edges are moves. It is a type of Markov Decision Process (S, A, R, δ, γ) wherein S - State, A - Action, R - a Reward Function (S x A), δ - a Transition function and γ - a Discount Factor.

The most naive way to make an AI would be for it to try list out and examine every possible valid move it can make. This examination would for each possible AI first move, require examining its human opponents next possible second move. Which in turn would require examining move 3 by the AI. And so on until we reach an end state. Game

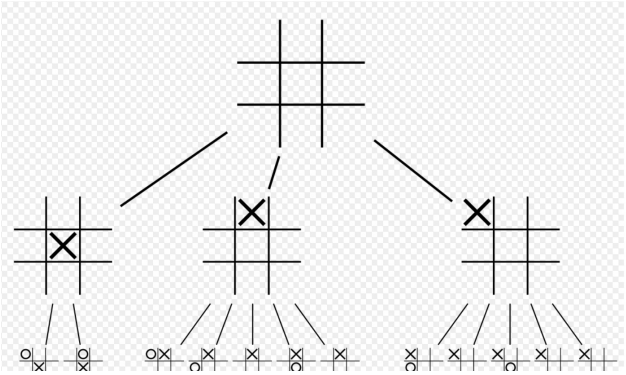


Fig. 1. Sample game tree of tic-tac-toe, source: wikimedia.org

complexity encapsulates state-space complexity, game tree size, decision complexity and computational complexity. As the branching factor increases, the difficulty of the problem scales up exponentially. The branching factor for a game can be considered as the out-degree or the number of children in each node. The complexity number(10 to the power of) denotes the number of valid states possible. The following table contains details for three popular board games.

TABLE I
Game Complexity

Sl No	Game	Complexity	Branching Factor
1	Chess	47	35
2	Go	170	250
3	Gomoku	70	210

The AI Chess program Deep Blue defeated world champion Garry Kasparov in 1996. As can be seen in the table I above, branching factor of Go is about 8 times higher when compared to chess (35 vs 250). Consequently, AI developers at Google Deepmind took a further 20 years to be able to develop AlphaGo using deep neural networks and reinforcement learning [7], the first Go AI program to defeat a professional level Go player in late 2016.

This paper explores ways of optimizing a Monte-Carlo Tree Search (MCTS) method with Genetic Algorithms(GA). This paper is structured as follows. Section 2 and 3 provides an

978-1-5386-4966-4/18/$31.00 ©2018 IEEE

overview of the problem at hand and why gomoku was chosen for consideration.Section 4 gives a brief overview on Genetic Algorithms, and Section 5 on the improvements that will significantly speed up the AI.

II. Gomoku or Connect-5

Gomoku is played with black and white stones on a 15x15 grid. For simplicity, it can be described as an m,n,k problem (e.g tic-tac-toe). It is a a 2 player, zero sum, perfect information board game requiring skill. The first player to get 5 stones in a row wins.

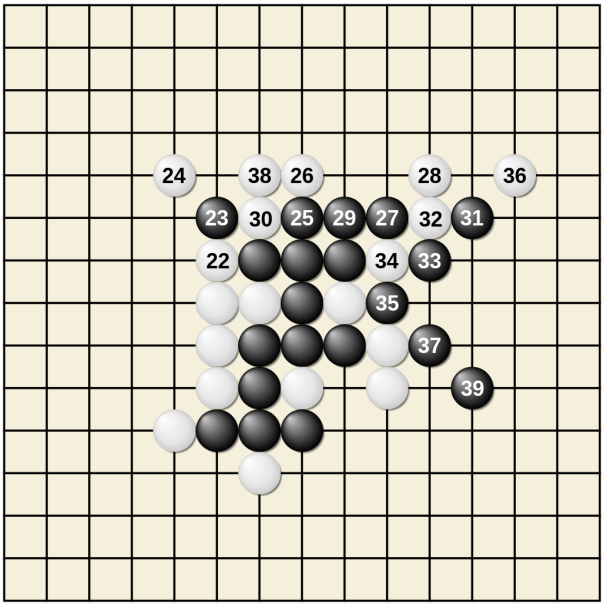


Fig. 2. Sample game of Gomoku, source: wikipedia

A number of variants of Gomoku exist, the primary aim being to balance the game between the first and second player. Standard gomoku offers a huge advantage to the first player (playing black). Louis Victor Allis [1] proved that, with a perfect gameplay, the first player will always win. The most accepted variant now is the Gomoku-swap2. A The first player puts three stones (two blacks and one white) on any intersections of the gomoku board. The second player has three options now. One, the player can choose white and put the 4th stone. Two, the player can swap colors and control the black stone. Three, the player can put two more stones (one black and one white stone, to make it five stones on the board) and then let the opponent choose the color.

III. MCTS - Monte Carlo Tree Search

MCTS is based on the proof-number search (p-n search). Dr L.V Allis [1] had introduced the proof-number search (p-n search) method in his doctoral thesis in 1994. The game tree of a two-person, zero-sum, perfect information game is mapped to an and-or tree. A proof number represents the minimum number of leaf nodes which have to be proved in order to prove the node. A disproof number represents the minimum number of leaves which have to be disproved in order to disprove the node. For all nodes proof and disproof numbers are stored, and updated during the search. Because the goal of the tree is to prove a forced win, winning nodes are regarded as proved. Therefore, they have proof number 0 and disproof number . The proof number of an internal AND node is equal to the sum of its children's proof numbers, since to prove an AND node all the children have to be proved. The disproof number of an AND node is equal to the minimum of its children's disproof numbers. The disproof number of an internal OR node is equal to the sum of its children's disproof numbers, since to disprove an OR node all the children have to be disproved.

Monte Carlo tree search (MCTS) combines the precision of tree search with the generality of random sampling [4]. It has received considerable interest lately due to its achievements in solving problems of considerable difficulty like Go. The basic idea of MCTS is to view the action selection in a state as a multiarmed bandit problem and sample the outcome in a Monte Carlo fashion. MCTS is ideal both deterministic as well as stochastic games where heuristic search meets its limits. It has also proved beneficial in a range of other domains.

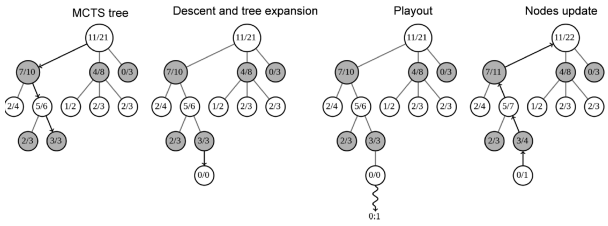


Fig. 3. Stages of Monte Carlo Tree Search [8]

Figure above shows the 4 stages of MCTS. Firstly, a selection strategy is applied until a leaf node is reached. The stored tree is then expanded by adding a new node to this leaf. Its quality is estimated by performing a rollout and propagating the achieved outcome as an estimate for all nodes among the selected path. Those nodes store the number of updates and the average estimate. This storage allows to trade off exploration of new nodes with exploitation of already discovered promising nodes [5]. Practically, depth limits are used if it is too expensive to compute rollouts till a terminal state. A benefit of MCTS is that the values of intermediate states do not have to be evaluated (as in minimax search). Only the value of the terminal node at the end of each simulation is required.

Searching game-tree always brings up the previously mentioned question of exploration vs exploitation. I.e upon finding a suitable node, should resources be spend to search for a better node; or is it better to repeatedly keep on selecting this node itself. This is analogous to the problem in which a gambler at a row of slot machines has to decide how many times to play each machine. The Upper Confidence Bounds (UCB) value is an ideal and commonly used method in an AI agent to balance exploration vs exploitation. The UCB value is calculated as vi + C * sqrt(ln N / ni), where vi is the estimate value, ni is the number of times the node has been visited, N

is the total number of times its parent has been visited and C is a tunable bias parameter.

IV. Solving Gomoku using G.A

Genetic Algorithms can be used to search the game tree search space. Each possible gameplay can be considered as a chromosome, with a set of parameters representing each move in the game. Thus, a set of n chromosomes can represent all cases possible in a perfect information game like Gomoku. The usage of mutation and crossover operators mean that not all scenarios in the search space need to be iterated. Junru Wang et al [3] have described this scenario. The G.A approach is divided into the following steps.

Step 1 - Individuals are generated randomly, but confirming to the rules of Gomoku.

Step 2 - Population is evaluated by a fitness function. Fitness function can vary. Wang et al have used a pattern matching approach to find winnable patterns. Fitness functions that check the number of symbols in a row are also adept. Low value chromosomes are discarded.

Step 3 - Evolution is done by mutation and crossover operators.

Step 4 - The above steps 1 to 3 are repeated till there is a convergence. I.e, we have a solution with a win possibility. The fitness function can be considered as a measure of quality, of how much the problem is solved by this particular solution. The possible layouts are listed into 12 types as shown below along with their values.

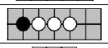
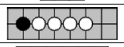
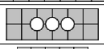
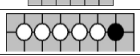
ID	Pattern	Value
1		1
2		3
3		7
4		15
5		31
6		63
7		127
8		255
9		511
10		511
11		511
12		1023

Fig. 4. Layout Patterns and their values

V. Algorithm Improvement

A. Search Space Reduction

GA algorithm's fundamental idea is to generates a population using randomized moves. But this can be clearly improved by using a dynamic search space reduction. Gomoku is a game wherein pieces once placed on the board are not removed in any circumstances. This is unlike games like GO, wherein opponent pieces can be 'captured' by surrounding them and then removed from the board [7]. This behavior can be used to significantly improve the AI stratagem. Consider the below sample scenario, a game in progress wherein the AI controls 'X' and has to make the next move.

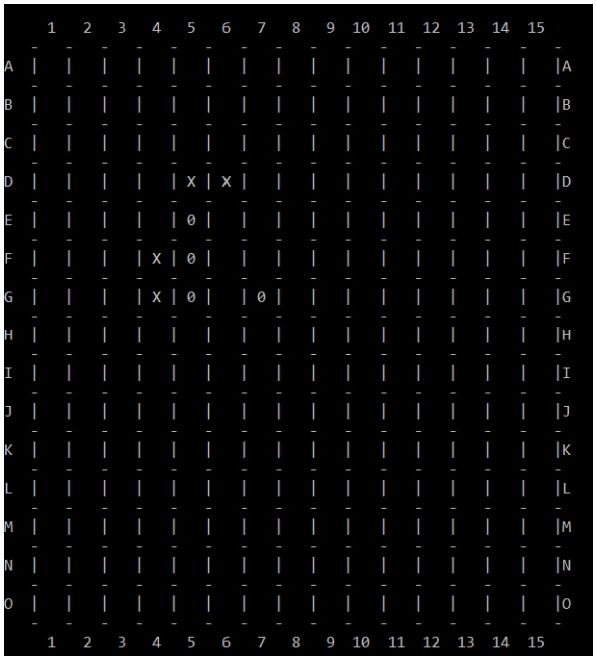


Fig. 5. Gomoku game in progress

In the above scenario, AI does not have to consider all available legal moves (blank spaces from A-1 to O-15). Instead, it is enough if AI considers blank spaces from C-3 to H-8. This is reduction is obtained by using a top, bottom, left and right cutoff. The left cutoff is obtained by finding the leftmost non-blank cell in the grid and subtracting 1 from its column number. Here its F-4. So it is enough if cells from the 3rd column onwards are considered. This follows the principle that all moves made must be to try reach a victory outcome. This outcome can be achieved only by maximizing player chances (attacking, try winning) or minimizing opponent chances(defending, prevent opponent win). This can be accomplished only by playing cells close to one's pieces(attacking) or opponent pieces(defending).

B. Clustering

It is to note that the ideal AI should have a small bias towards attack over defense. This is to try to decisively converge to an ideal win state. Also, if two cells are similar in positional play, the cell with a larger number of blanks will have a priority, as each blank cell next to AI move represents a possibility in future that the adjacent cell can be played to AI's advantage. The above two ideas can be used to assign a fitness number to a blank cell. Its fitness can be considered

as the *number of 'X' in its immediate neighbour cells * $\alpha 1$ + number of '0' * $\beta 1$ + number of blanks * $\gamma 1$*. The $\alpha 1$ value represents the relevance of 'X' in the adjacent cells, $\beta 1$ for '0' and $\gamma 1$ for blank space. For now, only first level $\alpha 1$, $\beta 1$ and γ values are considered and they are taken as below based on intuition. Similarly $\alpha 2$, $\beta 2$ etc values can be considered for second level relevance, i.e, at a distance of two cells from the current move.

TABLE II
ADJACENCY WEIGHTS

Adjacency No	Weights		
	α	β	γ
1	1.1	1	0.1
2	$\alpha 2$	$\beta 2$	$\gamma 2$
3	$\alpha 3$	$\beta 3$	$\gamma 3$

The values in table II can be calculated experimentally by creating two separate AI's with different values and playing them against each other. Note that in this case, the AI's are simple, with the algorithm consisting of only clustering. Keeping one AI's value constant and incrementing the α, β and γ values of the second AI in small steps will lead to convergence to an ideal value. Converge is indicated by the second AI winning more games after the change is done. Similarly $\alpha 2$, $\beta 2$ etc values can be calculated.

The above idea can be applied to the earlier game in progress in Fig 5 as shown below. Table III indicates the cell id, the number of 'X', '0' and blank spaces and the weight calculated.

TABLE III
MOVE ANALYSIS

Cell Id	X	O	Blank	Total Wt
D-3	0	0	8	0.8
D-4	1	1	6	2.7
E-4	2	2	4	**4.6**
E-6	2	2	4	4.6
F-6	0	4	3	4.3
G-6	0	3	5	3.5

C. Relevance Zone

The result in table III is used to select a list of cells is considered in the next level. Cells with value greater than 1.2 are chosen. (1.2 value being a corner cell with 2 blank spaces.) This includes cells D-4, E-4 etc. But this can be further optimized. Since clustering is used, the algorithm returns E-4 as the cell with the highest value. But upon examination of table III, it is found that the cell D-4 is best as it leads to a sure victory. Refer fig 4. Playing D-4 would create *two* 3 in a row scenario, one of which is guaranteed to become 4 in a row scenario. Such a scenario (wherein D-4 is not selected above E-4) can be avoided by adding the concept of relevance zones.

The concept as such was introduced by Shi-Jim Yen et al [2] in a new MCTS variant called Kavalan for playing Connect6. Their approach used a two-stage MCTS. The first stage focuses on threat space search (TSS), which is designed to solve the sudden-death problem (the problem when an immediate move is the best one leading to victory straightaway, but it is not found out by the heuristic search). Threat Space Search (TSS) - uses the concept of threat (defined in L.V Allis [1] paper) and aims to generate a larger number of threats than the number of legal stones the defender can play. This ensures the defender cannot block victory for the attacker. A double-threat TSS is introduced wherein the player finds double-threat moves. A double move generation is done by using connection patterns by the defender to block double-threat moves. This kind of defense with double threat TSS is called a conservative threat space search (CTSS). An algorithm called iterative threat space search (ITSS) is proposed in the paper, which combines normal TSS with conservative threat space search (CTSS).

Relevance Zone is used by counting the number of 'X', '0' and blank spaces in all rows, columns and diagonals. Consider the example of the min phase (AI trying to win with 'X') in Fig 5. Here the cell D-4 is the intersection of row D and column 4, both of which have count of 'X' as 2. If D-4 is a blank cell, it is identified as a good candidate and considered for the next phase by incrementing their weight. Thus the relevant zones are used to enhance the algorithm.

VI. PLAYOUT WORKING

The above algorithm is combined with a min-max approach as below. Consider the case wherein AI is playing as 'X' and has the next move. AI checks the below. Step 1: Can AI win in 1 move? This is the min state. AI checks blank spaces after space reduction. No further checking is required as this is the ideal state. Step 2: Can AI prevent user winning in move 2? This is the max state. Checking is similar to Step 1, except that 'X' is now reversed to '0'. Step 3: Can AI win in move no 3? Step 4: Can AI prevent player winning in move no 4? Step 5: Can AI win in move no 5? and so on...

Steps 3 onwards are checked by the algorithm. It creates each state it checks (the chromosome) by adding the list of ideal moves by both the AI as well as the user player. This proceeds upto depth n(depending on computation resources available) or until a victory state is returned. Thus the path with the shortest move to the victory state(if any) is chosen by the AI as its next move.

The same algorithm optimization can be easily integrated into the previously mentioned G.A approach as follows. Step 1 - Individuals are still randomly generated but after a search space reduction. This considerably increases the speed of convergence to a solution. Step 2 - The Fitness function is considerably improved by adding the concept of clustering and relevance zones.

VII. CONCLUSION

MCTS offers significant advantages over other methods like alpha-beta pruning [6]. The flexibility of MCTS allows it to be hybridized with a range of other techniques. MCTS can be

employed to any problem just by having the state, transitions and win conditions; i.e effective gameplay can be obtained with no knowledge of a game beyond its rules. MCTS can be further enhanced easily by incorporating human knowledge, machine learning or other heuristic approaches. MCTS tree search reduces branching factor and concentrates on more promising sub-trees, thus results are much better than classical algorithms.

A disadvantage is that, in cases where a single branch leads to a single loss node, MCTS may not always find it. Because this is not easily found at random, the search may not take it into account. For video game and real-time control applications, a systematic way to incorporate knowledge is required in order to restrict the sub-tree to be searched.

REFERENCES

[1] Louis Victor Allis, "Searching for Solutions in Games and Artificial Intelligence," Doctoral Thesis, University of Limburg, Maastricht, The Netherlands.

[2] Shi-Jim Yen and Jung-Kuei Yang, "Two-Stage Monte Carlo Tree Search for Connect6," IEEE Transactions on Computational Intelligence and AI in Games, Vol. 3, No. 2, June 2011.

[3] Junru Wang and Lan Huangb, "Evolving Gomoku Solver by Genetic Algorithm," 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA).

[4] Cameron B. Browne and Edward Powley, "A Survey of Monte Carlo Tree Search Methods," IEEE Transactions on Computational Intelligence and AI in Games, Vol. 4, No. 1, March 2012.

[5] Hendrik Baier and Mark H.M Winands, "MCTS-Minimax Hybrids," IEEE Transactions on Computational Intelligence and AI in Games, Vol. 7, No. 2, June 2015.

[6] Hendrik Johannes Sebastian Baier and Dr. L.L.G Soete, "Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains," Doctoral Thesis, Maastricht University, The Netherlands, 2015.

[7] D Silver, A Huang, C J Maddison, A Guez, L Sifre, G van den Driessche, J Schrittwieser, I Antonoglou, V Panneershelvam, M Lanctot, S Dieleman, D Grewe, J Nham, N Kalchbrenner, I Sutskever, T Graepel, T Lillicrap, M Leach, K Kavukcuoglu, D Hassabis, "Mastering the game of Go with deep neural networks and tree search," Nature(journal), Jan 2016.

[8] Amit Benbassat and Moshe Sipper, "EvoMCTS: A Scalable Approach for General Game Learning," IEEE Transactions on Computational Intelligence and AI in Games, Vol. 6, No. 4, Deccember 2014.