Application of Monte Carlo Search Tree in AlphaGo

Wenny Yustalim 13515002
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515002@std.stei.itb.ac.id

***Abstract*—Playing a combinatorial game might sometimes be extremely difficult for the human mind. For centuries, humans have always attempted to develop algorithms in order to discover a way to always win every match of a particular game, which is not only efficient in time, but also has almost 100% chance of winning. One of the examples of this combinatorial game is "Go", a Chinese board game similar to chess, but much more complex and mindblowing. This paper discusses how the Monte Carlo tree search is used in the computer program AlphaGo in order to win almost every match of a Go game.**

***Keywords*—AlphaGo, Go, Monte Carlo tree search, Zhuo Zhuan.**

I. INTRODUCTION

Go (圍棋 *wéiqí*) is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent. The game was invented in ancient China more than 5,500 years ago, and is thus the oldest board game continuously played today. The earliest written reference to the game is generally recognized as the historical *Zuo Zhuan*.

The game is simply played by two players taking turns to place black or white stones on the board. The player must try to capture the opponent's stones or surround empty space to make points of territory.

The stones are placed on a (traditionally wooden) board of size 19x19. There are also smaller sized boards, 9x9 and 13x13, but the standard board size for tournaments is the 19x19 one. Due to this reason, the possible moves that the players can make is much more than the moves in a chess game, since the size of the board is larger than the standard 8x8 sized chessboard, and the players can place the stone almost anywhere on the board, except for some restricted areas that will be explained further.

The empty points which are horizontally and vertically adjacent to a stone, or a solidly connected string of stones, are known as liberties. An isolated stone or solidly connected string of stones is captured when all of its liberties are occupied by enemy stones.

A player may not self-capture, that is play a stone into a position where it would have no liberties or form part of a string which would thereby have no liberties, unless, as a result, one or more of the stones surrounding it is captured.

Any string or group of stones which has two or more eyes is permanently safe from capture and is referred to as a live string or live group. Conversely, a string of stones which is unable to make two eyes, and is cut off and surrounded by live enemy strings, is called a dead string since it is hopeless and unable to avoid eventual capture.

There is also a major difference between Go and chess. In Go, the players start with completely nothing on the board, hence they have to really consider where to place their first stone in order to gain maximum winning chance, while in chess, the pieces are already placed on both sides of the board, and the players have to move them around in order to win the game.[2]

There is also a rule called *ko* that prohibits repetition of any previous position, the ko rule prohibits only immediate repetition, that can lead the game into an infinite repetition and thus can take forever to finish.

As simple as the rules are, Go is a game of profound complexity. There are more possible positions in Go than there are atoms in the universe. That makes Go a googol (10^{100}) times more complex than chess.[2]

Go is played primarily through intuition and feel, and because of its beauty, subtlety and intellectual depth it has captured the human imagination for centuries. AlphaGo is the first computer program to ever beat a professional, human player.

AlphaGo is a computer program developed by Google DeepMind in London to play the board game Go. In October 2015, it became the first Computer Go program to beat a professional human Go player without handicaps on a full-sized 19×19 board. In March 2016, it beat Lee Sedol in a five-game match, the first time a computer Go program has beaten a 9-dan professional without handicaps.[6] Although it lost to Lee Sedol in the fourth game, Lee resigned the final game, giving a final score of 4 games to 1 in favour of AlphaGo. In recognition of beating Lee Sedol, AlphaGo was awarded an honorary 9-dan by the Korea Baduk Association.[3]

AlphaGo uses very much the application of Monte Carlo tree search programs that simulate thousands of random games of self-play. Using this search algorithm, AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. That is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away. [7]

It is very interesting that such a simple concept of search tree can be used to create the algorithm that can win over even the most skilled human player of Go. It took Google a lot of reasearch, time, energy, and funds in order to make AlphaGo a real thing that can actually match the brain of a human. It uses traditional AI methods such as alpha-beta pruning, tree traversal and heuristic search. The most dominant part is the Monte-carlo tree search, which is going to be explained in this paper.

The Monte-carlo tree search is a heuristic search algorithm for decision processes, that is nowadays used in many AI (Artificial Intelligence) in order to win a game, such as poker, Go, and many other examples. The focus is analysing the most promising moves, expanding the search tree based on random sampling of the search space. The game is played to the very end by selecting random moves, then the final result is used to weigh the nodes in the tree so that better nodes will be used more often in the next games. This method allows AIs to learn which moves produce the best possible outcome.

There is another algorithm called Minimax to search on a tree, by considering several factors into account such as picking the maximum and minimum value, matching different players' conflicting goals, and it proceeds down the tree. This tree basically finds the best standing that will result in the best outcomes possible. It contains all possible outcomes. But this algorithm is not the most efficient one so that AlphaGo prefers to use Monte-carlo tree search instead. The problem with this algorithm is that it can take a considerably long time to do a full search on the tree, especially for a game with high branching factor (a high average number of available moves per turn).

II. THEORIES

2.1 Game Tree

To understand how AIs are capable of playing games such as chess and Go, we have to understand what a game tree is. A game tree represents game states (positions) as nodes in the tree, and possible actions as edges. The root of the tree represents the state at the beginning of the game. The next level represents the possible states after the first move, etc. For simple games such as tic-tac-toe, it is possible to represent all possible game states (the complete game tree) visually:

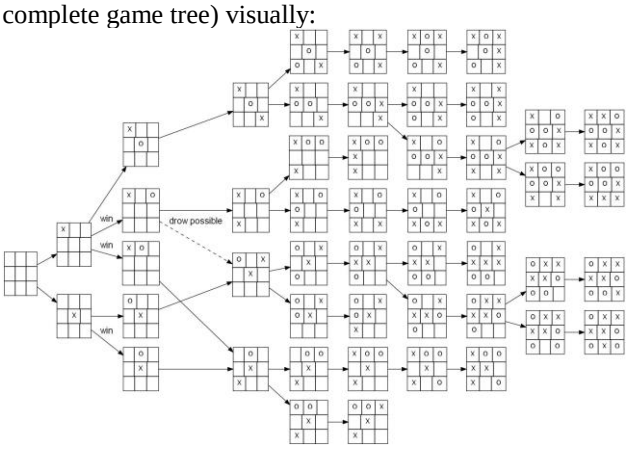

Figure 2.1 Game tree of a tic-tac-toe
(Source: https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/)

Knowing the complete game tree is useful for a game playing AI, because it allows the program to pick the best possible move at a given game state. This can be done with the minimax algorithm: At each game turn, the AI figures out which move would minimize the worst-case scenario. To do that, it finds the node in the tree corresponding to the current state of the game. It then picks the action that minimizes the worst possible loss it might suffer. This requires traversing the whole game tree down to nodes representing end-of-game states. The minimax algorithm therefore requires the complete game tree. Great for tic-tac-toe, but not useful for chess, and even less so for Go.

2.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an alternative and a much more efficient approach towards searching the game tree. The idea is to run a certain number of game simulations. Each simulation starts at the current game state and stops when the game is won by one of the two players. At first, the actions from both players are chosen randomly. At each simulation, the values are stored, such as how often each node has been visited, and how often this has led to a win. The later simulations are guided by these recorded numbers, thus making the next simulations less and less random after a certain number of simulations. The actions that are taken in the future are more precise. The more simulations are executed, the more accurate these numbers become at selecting winning moves. It can be shown that as the number of simulations grows, MCTS indeed converges to optimal play.

Each round of Monte Carlo tree search consists of four steps:

1) Selection: start from root R and select successive child nodes down to a leaf node L. The section below says more about a way of choosing child nodes that lets the game tree expand towards most promising moves, which is the essence of Monte Carlo tree search.
2) Expansion: unless L ends the game with a win/loss for either player, either create one or more child nodes or choose from them node C.
3) Simulation: play a random playout from node C. This step is also called playout or rollout.
4) Backpropagation: use the result of the playout to update information in the nodes on the path from C to R.

Sample steps from one round are shown in the figure below. Each tree node stores the number of won/played playouts.

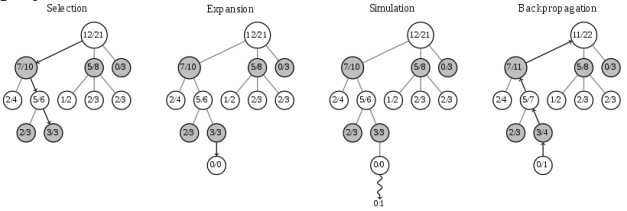

Figure 2.2 Steps of Monte-carlo Tree Search
(Source: https://upload.wikimedia.org/wikipedia/commons/)

2.3 AlphaGo

AlphaGo relies on two different components: A tree search procedure, and convolutional networks that *guide* the tree search procedure. The convolutional networks are conceptually somewhat similar to the evaluation function in Deep Blue, except that they are *learned* and not *designed*. The tree search procedure can be regarded as a brute-force approach, whereas the convolutional networks provide a level on intuition to the game-play.

In total, three convolutional networks are trained, of two different kinds: two *policy networks* and one *value network*. Both types of networks take as input the current game state, represented as an image.[8]

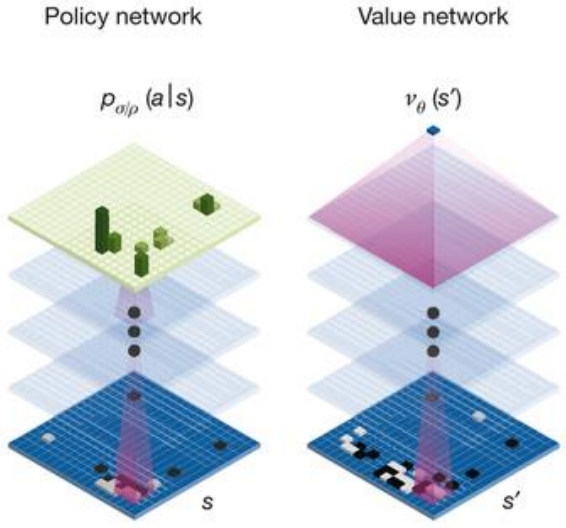

Figure 2.3 Types of Network in AlphaGo
(Source: https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/)

The value network provides an estimate of the value of the current state of the game: what is the probability of the black player to ultimately win the game, given the current state? The input to the value network is the whole game board, and the output is a single number, representing the probability of a win.[8]

The policy networks provide guidance regarding which action to choose, given the current state of the game. The output is a probability value for each possible legal move (i.e. the output of the network is as large as the board). Actions (moves) with higher probability values correspond to actions that have a higher chance of leading to a win.[8]

III. THE APPLICATION OF MONTE CARLO TREE SEARCH

3.1 The Purpose of using Monte-carlo Tree Search

One might think, how will I ever win by only playing random simuation all the time? The basic idea of the application of Monte-carlo tree search is that if I stand better on the board than my opponent, then although I play random moves all the time, the chances of winning is still better than when I stand worse than my opponent on the board.

This is not a humanlike way to play the game of Go. When I play the game of Go, I do not run simulations on my head, counting out the chances I have or figuring out the best possible stance I could possibly have then apply this to the tree search. That would probably take me hours, months, or even years to find the best spot to place my stone. But a good analogy to this problem is this: centuries ago, humans had not have any imagination that in the future, humans will be able to fly, but not with flappy wings like birds. We are now able to fly on planes, on helicopters, by the use of steel and heavy materials that have been very specifically tailored to carry passengers, with accurate calculations that are very complex to the human brain.

The Go game is very complicated that it is much more profitable that humans create the algorithm in order to do the thinking, instead of humans doing all the vigorous thinking all by ourselves. This is what machines can do better than us humans.

Another good concept as to why AlphaGo uses the Monte-carlo tree is the multi-armed bandit problem. It is a problem in which a gabmler at a row of slot machines has to decide which machines to play, how many times to play each machine, and in which order to play them. When played, each machine provides a random reward from a probability distribution specific to that machine. The objective of the gambler is to maximize the sum of rewards earned through a sequence of lever pulls.

When someone tries to play the machines and fails several times, he will start to think that maybe the other machines will give him a jackpot. Then he will start playing the other slot machines and see which one produces a win for him. He wants to exploit and explore all the machines.

3.2 Brief Explanation on the working method

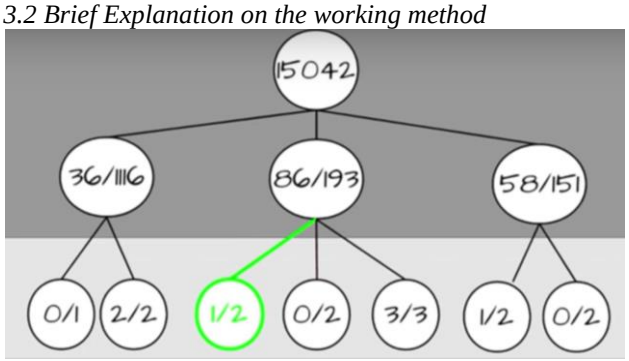

Figure 3.1 Example of Monte-carlo Tree Search
(Source: https://www.youtube.com/watch?v=b9H9AtbxpPM)

The picture above shows a tiny part of the whole tree. The root shows the number of times the node has been visited. The child nodes show how much wins per how much times the nodes have been visited. For example, 86/193 shows that the node has been visited 193 times, and the path that has passed through that node until the leaf has resulted in 86 wins. After the 86/193 node, the ½ node is visited. This is chosen randomly. Even though 3/3 has the higher chance of winning (100%), but this is still the simulation part of the tree search, where the path is chosen randomly to find whether even the ½ node can afterall result in the higher stance. It is possible that after hundreds and thousands of simulation, this node will

result in a higher winning percentage than the 3/3 node, the 1 loss was probably just *unlucky*.

This step is done over and over again, and the only thing that matters in this part is whether or not when the search has ended up on the leaf, the game is won or not. If it is won, then the information is accumulated and analyzed.

The program can be stopped anytime, and can be asked to show which move is the best one in a particular state of the game. This is similar to using GDB (GNU Debugger) for debugging programs, where you can run specific line of codes and put breakpoints anywhere, look at the registers, and do miraculous things in order to find where the bugs hide.

This Monte-carlo tree search is sometimes considered as a *lazy* way. The only thing it cares is about winning the game. It does not care about how much points the players get, how far the gap between the players' states is, etc. It only cares about winning.

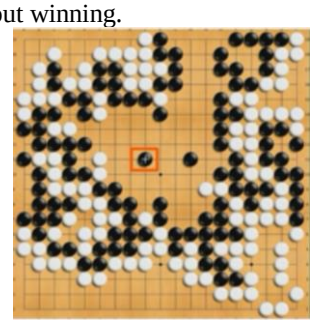

Figure 3.2 Game state that shows AlphaGo's laziness (Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

For example, in the picture above, the program puts the black stone straight in the middle of the board, even though there are some dangers on the stones around the perimeter. Almost no human would put the stone on that spot. But the program thinks differently than us. The centre was assumed to be safe and sound. The computer knows better than us. It was probably thinking that on the game state, it was 2.5 points ahead, if it puts the stone in the middle, it will become only 1.5 points ahead, but the chance of losing is very slim.

Each and every small move that is decided by a stone placed on the board creates a global impact of moves. The result of the game can be very much decided even by the first move that you made.

In the large board of 19x19, almost every move is legal. What makes the AlphaGo different from chess is that the moves in chess are pretty much restricted to a certain pattern, meanwhile in Go, the player is almost free to place the stone anywhere his heart desires. The average branching factor of chess in 35, whereas in AlphaGo it is 250. This means that for each and every stone placed on the board, it creates 250 other possibilities of states that can be reached. The state space complexity of AlphaGo is 10^{171} compared to chess at the number of 10^{47}.

3.3 The Mechanism of the Tree Search

As it has been explained on the introduction, the Monte-carlo Tree Search takes 4 steps: selection, expansion, simulation, backpropagation.

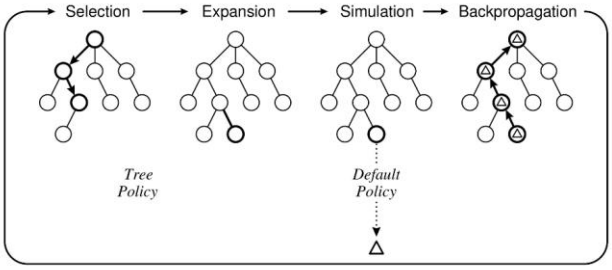

Figure 3.3 Steps through a Monte-carlo Tree Search (Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

This shows that on the selection step, it will randomly select on which branch the tree will be accessed, then it will go to the expansion step and check whether the node needed already exists, if the node does not exist previously, it will create a new node, thus expanding the tree downwards. Then it will go to the simulation step, where the tree will be accessed even deeper and further in order to check the result of the decisions. Then the result is propagated back to the root. If the path results in a win, then it will be considered as having a higher chance of winning, and will be explored further in the next simulations. This process is repeated over and over again by AlphaGo until it has almost accurate data over the possibilities of a game of Go.

3.4 The Similarity to the Neural Networks

Neural networks are a computational approach based on large collection of neural unites loosely modeling the way our biological brain solves problems. Each neural unit is connected with many others, and can affect the activation state of connected neural units.

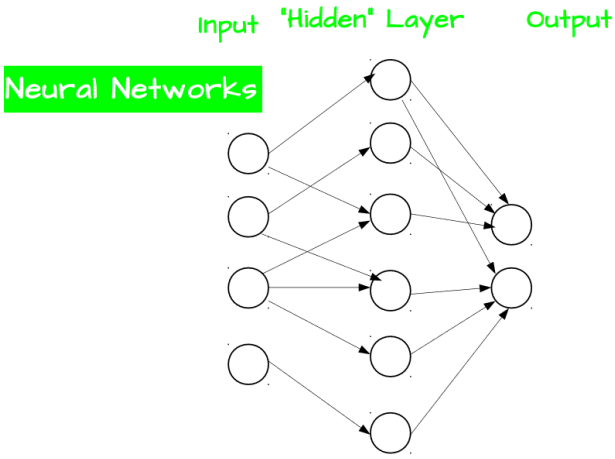

Figure 3.4 The Neural Network illustration (Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

The neural networks has input, hidden layer, and output. The hidden layer is the complex way of processing inputs into outputs. It also has weights and biases/thresholds. The input is basically the 19x19 board that is processed into the output. The bias tells whether it is going to activate the next layers.

How does the neural network learn? It learns from training. The training adjusts the parameters, weights and

biases through supervised learning. We have to have a clear data set, get input from the dataset, and create the expected output, then from backpropagation to adjust these parameters.

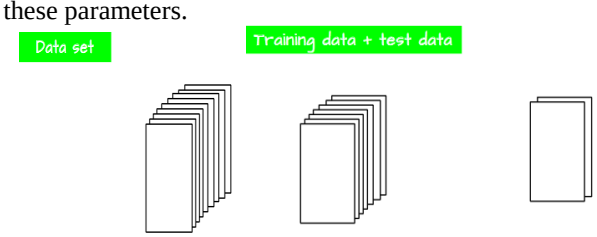

Figure 3.5 The separation of dataset
(Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

The dataset that has been collected is then divided into training data and test data. This is because when we only have the training data, then it does not imply a higher abstract level of data. This will prevent overfitting.

The deep neural network is even more abstract than neural network, with much larger input data.

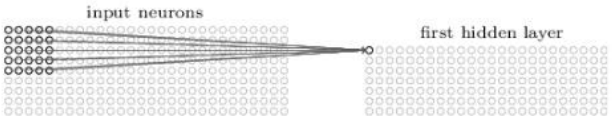

Figure 3.6 Convolutional Neural Networks
(Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

It takes all the input from the layer before it then maps it into the other field, then it moves along in a stride. The local receptive field will recognize the same or similar features by the shared weight and biases.

In order to shorten the amount of time needed, we have to minimize the parameters. AlphaGo trained it for weeks nonstop, now it has 2.3 million parameters and 630 million connections.

The input features are:

- Stone colour
- Liberties
- Liberties after move played
- Legal move
- Turns since
- Capture Size
- Ladder Move
- KGS Rank

This Deep Neural Network method may probably defeat GnuGo, but has only 55% of accuracy. But this is defeated by the Monte-carlo tree search. When they combine it with Monte-carlo tree search was much better than only the deep neural network.

3.5 The AlphaGo System

This system still has a flaw. It uses the ability of machine learning, as if it is a real human that is learning something. But if it is only developed in this direction, the program will only be able to match the ability of humans, but never surpasses it. It only imitates the human brain.

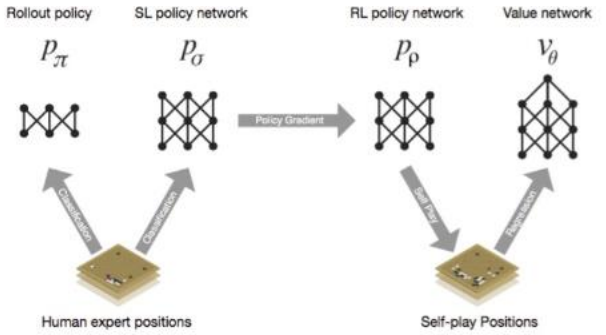

Figure 3.7 The mechanism of AlphaGo
(Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

What makes this special is that they create a self-play system where they play against itself all the time, it challenges itself all the time. That self-play gets better and better over time. AlphaGo never sleeps. It keeps learning how it can get better, how it can get even higher accuracy. It calculates the possibility of winning and losing.

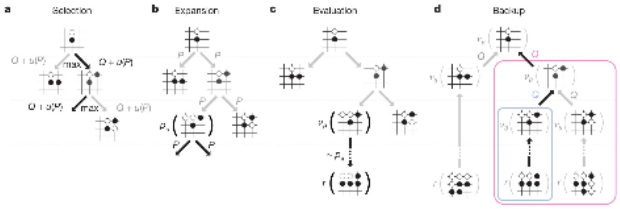

Figure 3.8 Combining AlphaGo with Monte-carlo Tree Search
(Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

It is then combined with the Monte-carlo tree search as the picture shown above.

The selection part is divided into 3 steps: action value, prior probability, and visit count. The action value is basically the rollout of the Monte-carlo search tree combined with other factors. The prior probability is the value they get from the superviced learning policy network. The visit count is how many times it has been visited. The more visit it has the less impact the prior probability has on that node being selected.

After the selection process, the program consult the value network, whether it looks like a winning or losing position to us. Then it carries out the backpropagation.

AlphaGo has 3 key strengths compared to human:

a) Policy Network x human instinct
Which tells us where we are likely to play

b) Search x Reading Capability
The tree search that can predict the many moves ahead

c) Value Network x Positional Judgement
The key factor to AlphaGo's success that tells us whether we are winning or losing.

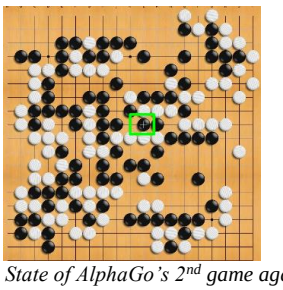

Figure 3.9 Game State of AlphaGo's 2nd game against Lee Sedol (Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

In the game against Lee Sedol, the world chamipon of Go, the green square was the last move made by AlphaGo. People thought it was a strange move, they thought that AlphaGo will surely lose. This is one of the *lazy* principle of AlphaGo. Humans will always think of the most profitable position in order to get closer to winning. We may regard that move as a mistake, but perhaps it should be more accurately be viewed as a declaration of victory. But AlphaGo thinks differently, it does not care about how much gap there will be, it only cares about winning the game. In the end, AlphaGo still wins.

Figure 3.10 Graph of confidence: AlphaGo vs Lee Sedol, 2nd game (Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

The green arrow shows the level of AlphaGo's confidence whether it is winning or not, in that particular move shown in the previous picture. It already knows that it is winning so it placed the stone there anyway.

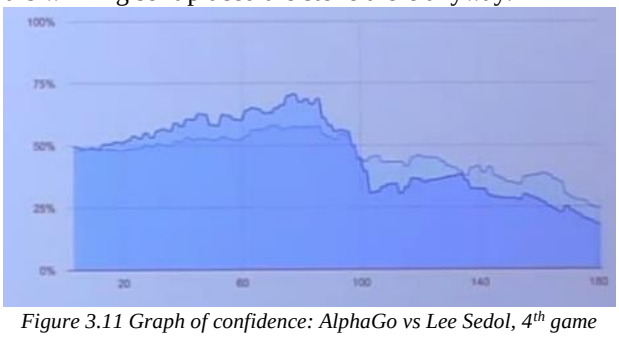

Figure 3.11 Graph of confidence: AlphaGo vs Lee Sedol, 4th game (Source: https://pragtob.files.wordpress.com/2016/09/strange_group.pdf)

This is where AlphaGo lost. Sedol made a defying all-out move in the middle of the game, and AlphaGo thought that it can still win with a tight margin. But when the graph started to go down, only then did AlphaGo realize that it was not a wise move and that it was losing. This proves that AlphaGo still needs a lot of improvement for its lack of knowledge.

V. Conclusion

Solving problems the human way is hugely different than solving problems the computer way. Sometimes we think that we might have to solve problems the same way we think, but then it is not correct because the human mind can still be defied by the actual way of winning.

We should not blindly dismiss the approaches as infeasible. Don't always assume that somebody else out there has done our ideas, that others might always do it better. If we have a problem and we want so desperately to solve it in a way better than the others, than we should try it. There may already be preceding solutions to the problem, but like the neural network way, it can be improved by the Monte-carlo tree search in order to increase its accuracy and efficiency, even exponentially higher than before. Some ways may even be complementing towards the previous ones and thus create an invention that the world has never seen before.

VI. Appendix

Convolution: a coil or twist, especially one of many
Googol: 1.0 x 10^{100}

VII. Acknowledgment

A special note of thanks to Dr. Ir. Rinaldi Munir, MT., one of the most inspiring lecturers at Bandung Institute of Technology for this interesting assignment that has broaden my knowledge on artificial intelligence.

I would also like to thank my friend Irene Edria that helped motivate me in doing this work, and for waking me up whenever I fell asleep while researching and writing this paper.

References

[1] W. Burton, "The Tso Chuan (reprint ed.)," April 1992.
[2] https://deepmind.com/research/alphago/ Retrieved 4 December 2016.
[3] *Artificial intelligence: Google's AlphaGo beats Go master Lee Sedol". BBC News. Retrieved 17 March 2016.*
[4] *"Research Blog: AlphaGo: Mastering the ancient game of Go with Machine Learning". Google Research Blog. 27 January 2016.*
[5] *"Google achieves AI 'breakthrough' by beating Go champion". BBC News. 27 January 2016.*
[6] *"Match 1 - Google DeepMind Challenge Match: Lee Sedol vs AlphaGo". 8 March 2016.*
[7] http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html Retrieved 4 December 2016. Retrieved 4 December 2016.
[8] https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/ Retrieved 4 December 2016.
[9] G.M.J.B. Chaslot; M.H.M. Winands; J.W.H.M. Uiterwijk; H.J. van den Herik; B. Bouzy (2008). "Progressive Strategies for Monte-Carlo Tree Search". IEEE Trans. Antennas Propagat., to be published.
[10] https://pragtob.files.wordpress.com/2016/09/strange_group.pdf Retrieved 9 December 2016.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Desember 2016

Wenny Yustalim 13515002