Go - Project

Jude Haris

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — May 2019

Chapter 1

Introduction

The ancient game of Go is regarded as the oldest board game still being played in the modern day. The origins of Go are not completely known but it is said to be invented in China 3000-4000 years ago. The game is formed by very simple rules that is played out most commonly in a 19x19 board. Each player takes turns placing a black or white stone on the intersections of the grid lines on the board. The board size is very crucial for the project. Due to increased board size compared to most western board games, a computer will take significantly more time to process through each possible move in every turn of the game to determine the final outcome. On the other side the basic principles of the game are much simpler to program and execute in software. Some complicated rules such as self-capture, and ko rule do exist. While implementing these rules can be simple,the effect of them being implemented correctly is great. For instance without implementing the ko rule it would be impossible to make progress in the game search tree in certain situations where moves could be played so that a never ending battle for the same point on the board takes place. Brute force methods to solve board positions would get stuck in these situation and loop infinitely.

1.1   Motivation

The simplicity of the game's rules are deceiving as the game's strategy and tactics used are very complex. The possible amount of different games of Go to occur is inconceivably large. It has been said to learn the game it takes little time but to become an expert or remotely good at it takes years maybe even decades. To allow a computer to calculate the perfect move via brute force would take too long to be considered viable. The current best AI built to play Go is AlphaGo, it has defeated one of the world's best Go player Kie Jie in 2017 at Future of Go Summit. AlphaGo uses Monte Carlo Tree Search used with neural networks to evaluate the board and calculate the best move. With that in mind AlphaGo is yet to be anywhere near perfectly solving Go due to the enormous complexity and possibility of moves.

1.2   Aim

The aim of this project was to create an AI to play against users in Go, which can choose the correct move given a board position. Specifically, the project's aim was to create a program which given a Go life and death problem it would solve it, if solvable or return as unsolvable. First milestone was to achieve perfect AI which would choose the correct move given infinite time or smaller problem with less valid moves. The goal was to use brute force search method to look through every possible board state that could occur given the current state and choose the move which will lead to victory for the computer if there is any. The final goal of the project was

to alter and enhance the AI which was achieved through the first milestone to be able to solve larger problems in the same time scale. And to do so, use heuristics to determine how favourable a board position is after searching for certain depth into the possible line of choices and then picking the line which leads to the most favourable board position. This final aim was set so that more realistic problems could be solved using the final product as without larger problems would simply take too long to play through. The motivation behind the project was to create a tool for Go players to use to improve tactical skill in Go via an interactive problem solver which helps the user figure out the solutions to life and death problems by having the computer play against them.

1.3 Project Outline

Chapter 2

Background & Related Works

To progress through the project certain level of knowledge of the game is required. Here are the key points of research conducted during the project.

2.1 Go

To understand Go as normal player would was crucial in the progress of the project. Many simple questions needed to be answered about the game and how it is played. Also, basic concepts and terminology needed to be understood before tackling complex strategies which are built up from these basic concepts.

2.2 Go Rules

There are many varied rulesets for Go. When these rules sets are broken into rules used during play and rules used at the end of play. The rules during play changes slightly and in most cases, they don't change at all. Any changes of these rules do not affect the strategy of the game at all. Rules used at the end of play refers to scoring and determining the winner of the game, while differences in these rules can be important in normal play, when subjected to the aim of this project they become irrelevant due to dead or alive nature of life and death problems. For these reasons the rules used during play are defined within the project but rules for the end of the game are left out, instead rules to decide if a problem is solved are defined.

2.3 Go Terminology

2.4 Life and Death Problems

2.5 Alpha Go

Chapter 3

Requirements

The project requires a well-defined and clear list of requirements due to the possible scale of it. Time constraints and resources available had to be taken to consideration before further work on the project was done. More precisely there was a need for identifying key aspects of the goals of this project and put them into more concrete requirements, so they can be implemented accordingly to achieve the goals of the project. Also, to keep track and have a streamlined work flow throughout the project the following requirements and scope were set.

3.1 Functional Requirements

To begin with the project's first milestone's aim which is to create an AI to play against user in Go life and death problems using brute force search. To achieve this, certain key requirements needed to be met:

The basic Go board should be displayed on the screen in which users can place stones in turn through mouse clicks. It needs comply to all the rules of Go and should capture and remove stones from the Board automatically after each turn. The program must be able to generate all the valid moves that can be made each turn and highlight them on the screen. Furthermore, the program itself must let the user create Go problems within it. To do so, split the program into two modes of use, an editor mode and player mode.

Editor mode needs to permit the user the ability to place black and white stones where needed without needing to take turns to create the initial board state they want their problems to be in. The state of the board needs to be valid always such that rules of Go would allow the board position to occur in normal play for it be able to be created in editor mode. Captured stones must be removed from the board and also only one ko point exists on the board. Editor should be able to define the boundary of play area, specifically to be able to choose which points on the board are in play for the problem. Keystones should be place able via editor mode, the keystone/s's purpose is to identify the stone in which the problem revolves around. Users should also be able to select who to play first, and whether to capture or keep alive the keystone/s. Finally, the editor mode is required to let the user to create a text description of the problem and save the entire problem into a text file. The save files needs be human readable and the program should be able to load them back to either edit the problem or to play them. The player mode is required allow the user to play through problems. This means it must allow users to load problems in from save files and play them out against the computer. To achieve the goal of creating an AI to play the game, there needs to be some features within the program to help with debugging. The ability to reset the problem back to original state, undo or redo a move and to also disable or enable the AI when required must be implemented into player mode.

The final requirement to achieve the first milestone needs to be addressed. There is a need for the program to

contain a simple but effective Alpha-Beta pruning search algorithm to allow the correct move to be determined by the computer given board of stones and which colour's turn it is, so that the computer can make that move to allow that colour to win. To make things simpler when problems are created, keystone/s needs to be placed on the board to allow the algorithm to surround its evaluations according to the status of the keystone/s on the board. For example, if the current turn is white and there is/are black keystone/s on board then the AI run the search algorithm and would evaluate to choose the move which would lead for the black keystone/s to be captured.

To achieve final aim of the project which is to adapt the AI created in the first part of the project to solve larger and more complex problems the following critical requirements are needed:

To introduce heuristics to the brute force Alpha-Beta search algorithm the program should contain a specialised module which evaluate the state of the board of each colour and returns how good or bad it is. This is so that given depth cut off to the search tree is implemented, the algorithm can return a value more accurately predict what would have been the result if the search has continued deeper. The AI must also use a complex move generator which given set of valid moves it picks certain number of moves and orders them according to the likelihood of being a good move using heuristics. Then the AI searches through this list of moves before going through and checking all the other possible valid moves.

In addition, with these critical requirements some sub-requirements are needed to help with testing and to finalise the product. A set of Go problems are needed to be created and built into the product. This is to allow users to immediately try out the player mode without needing to create problems by themselves, will be especially be useful for players new to Go. Another feature needed to be implemented was a valid move checker, to return a list of all possible moves either to enforce the rules of the game to the users to play properly or to allow the AI playing against the human to have a way to generate a list of possible moves to search through.

With all the requirements of the project it had become evident the scope of the project could vary drastically. To create a perfect Go playing AI on full 19x19 board would be far from achievable given the time and resources put into the project. The main constraints to decide the scope were:

- Time given for AI to search and find the correct move?

- The number of given valid moves a problem contains which the AI should solve?

These two constraints are closely related to each other, more valid moves the AI is expected to handle the more time it should be allowed but with an upper bound set to allow users to interact and solve the problems without waiting too long. These were needed to be adjustable according to the feedback given by user testing. Hence the AI had to be built in way to allow for flexibility regarding these two parameters. To do so, the program gives the ability to choose what depth the AB pruning algorithm searches to before it cuts off to the user. This directly affect both run time of the algorithm and scale of the problem as bigger problems can be allowed with smaller cut off depth also would lead to less time required.

3.2 Non-Funtcional Requirements

3.3 Must/Should/Could/Won't have

Chapter 4

Design

4.1 Design Principles

4.2 Brute Force Search

4.2.1 Alpha-Beta Pruning Search Algorithm

4.2.2 Bad Move Removal

4.3 Heuristic Search

4.3.1 Board Evaluation

4.3.2 Pattern Searching

4.3.3 Move Generator

4.4 Problem Files

4.5 Modes

4.6 Overall Architecture

Chapter 5

Implementation

5.1   Play Mode

5.2   Edit Mode

5.3   Game Board

5.4   Heuristic Evaluators

Chapter 6

Evaluation

6.1 Unit Testing

6.2 Evaluation of Heuristics

6.2.1 Types of Problems

6.2.2 Success Rate

6.3 Beta Testing with Go Players

6.3.1 Questionaires

6.3.2 Interviews

Chapter 7

Conclusion

7.1 Summary

7.2 Future Works

7.3 Reflection