The Game of Go: An Ideal Environment for Capstone and Undergraduate Research Projects

Timothy Huang

Department of Mathematics and Computer Science

Middlebury College

Middlebury, VT 05753

huang@middlebury.edu

Abstract

In this paper, we discuss how Go, a strategy game widely played in Asia and other parts of the world, provides a rich, challenging environment for capstone and undergraduate research projects. We first describe the game itself and the characteristics that make it appropriate for more advanced undergraduate projects. We then discuss several projects that our students have pursued over the last three years, and we share observations that may be helpful to other computer science educators.

Categories & Subject Descriptors

K.3 [**Computers and Education**]: Computer and Information Science Education – *computer science education, curriculum.*

General Terms

Documentation, Design, Human Factors.

Keywords

Go, capstone projects, undergraduate research, strategy games, artificial intelligence.

1 Introduction

The idea of using games such as Connect-4, checkers, or backgammon to teach computer science is not new; many students enjoy playing games and find extra motivation to work on game-related projects. The purpose of this paper is two-fold: First, we argue that a less commonly used game – the game of Go – provides an especially rich, challenging environment for students to pursue capstone projects and learn about academic research in artificial intelligence. Second, we describe several Go-related projects and share observations that may be useful to other computer science educators.

Invented in Asia over 3000 years ago, *Go* (called *weiqi* in China and *baduk* in Korea) is one of the world's oldest strategy games. Although Go players in Europe and the United States number only about 100,000 and 20,000, respectively, Go players in the Far East number over 25 million; Go is even considered the national game of Japan. There are roughly 700 professional Go players in the world, and each year they compete in tournaments sponsored by Chinese, Japanese, and Korean companies for prizes worth millions of US dollars.

Like many other strategy games, the rules of Go are deceptively easy to learn, but the game itself is difficult to master. Unlike most other strategy games, however, the challenge of writing a strong computer Go program remains unsolved. Even the world's best Go programs, many of which have been in development for over a decade, play only at the level of advanced beginning human players.

Thus, the challenge of creating a computer Go program is a genuine academic research problem that undergraduate students can readily grasp. The excitement of working on a computer strategy game and the opportunities to interact with the broader computer Go community help make computer Go a wonderful environment for undergraduate research.

This paper is organized as follows: In Section 2, we provide a brief overview of the game play of Go and its computational challenges. In Section 3, we discuss why Go is especially well-suited for advanced study. In Section 4, we describe several projects undertaken by our students over the last three years. Finally, in Section 5, we summarize our experiences.

2 The Game of Go

In this section, we present a brief description of how Go is played. A comprehensive explanation of the rules can be found in [6,7]; our aim is primarily to give a sense of the goal of the game and the factors that make it both enjoyable to play and difficult to master. We also discuss the characteristics of Go that make it such a perplexing computational problem.

2.1 Playing Go

Go is a two-player game played with black and white stones on a board of 19 x 19 intersecting lines, though 13 x 13 and 9 x 9 boards are sometimes used. Starting with black, the players take

turns either placing one stone onto one of the empty intersections or passing. The goal is to acquire territory (empty intersections) by surrounding it with stones. Once placed, a stone does not move; however, blocks of stones can be surrounded and captured by the opposing player. The game ends when both players pass. The player who has surrounded the most points of territory wins the game. In tournament play, there are no draws because the white player receives 5.5 additional points (called *komi*) to compensate for playing second.

Stones placed horizontally or vertically adjacent to each other form a *block*. The empty intersections next to a block are its *liberties*. A block is captured when all of its liberties are occupied by enemy stones. Consider the following examples from Figure 1: In the bottom left corner, a block of white stones has surrounded 12 points of territory. Towards the bottom right, a block consisting of a single black stone is surrounded on three of four sides. White can capture this stone by playing at its last liberty, the intersection marked **A**. Above those stones, White can capture the block of two black stones by playing at the liberty marked **B**. If it is Black's turn, Black can help those stones escape by playing at **B** first and creating a resultant block with three liberties.

In the upper left corner, black can capture the white stones by playing at **C**, thereby fully surrounding the three stones. Normally, it is illegal to play a stone where it will not have any liberties. However, playing a black stone at **C** is allowed because it captures at least one of the white stones directly adjacent to **C**. In the upper right corner, the white block is *alive*, i.e., safe from capture, because black cannot play a stone at **D** and a stone at **E** simultaneously.
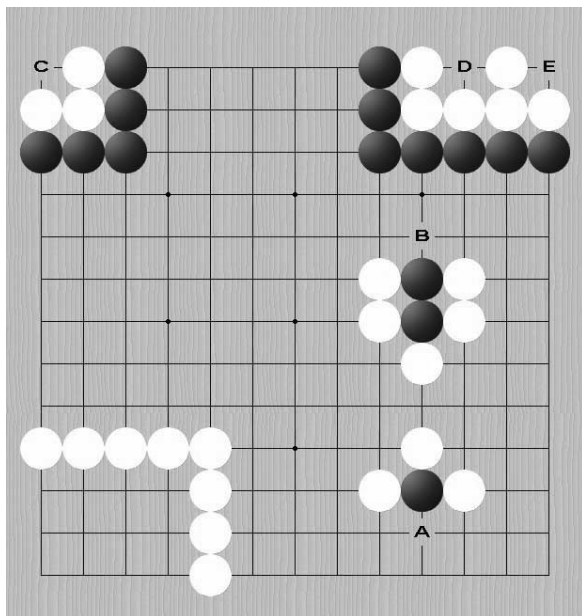

Figure 1: White can capture one black stone by playing at A or two stones by playing at B. Black can capture three white stones by playing at C. Since black cannot play at D and E simultaneously, the white stones in the upper right corner are alive.

The stones in Figure 1 were artificially arranged for explanatory purposes. Figure 2 shows a more realistic board situation partway through a game on a 13 x 13 board. In this game, white has staked out territory in the upper right, upper left, and lower left sections of the board. Meanwhile, black has staked out territory in the lower right and upper middle sections of the board and has pushed into the middle left section, thereby reducing white's territorial gain. Furthermore, black has effectively captured the white stones in the middle right section of the board (see [7] for a full explanation).

Part of what makes Go so engrossing and challenging is the interplay between tactics and strategy. On one hand, players fight localized battles for territory in small regions of the board. On the other hand, the arrangement of stones on one side of the board affects the strength and usefulness of stones on the other side of the board.
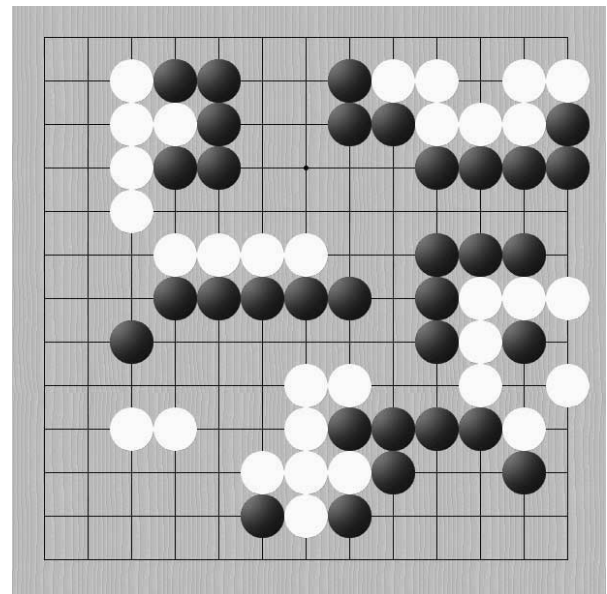
Figure 2: White's turn to move partway through an example game on a 13 x 13 board.

Figure 3 (in Section 4) shows the board situation at the end of the same game from Figure 2. Black's territorial points are marked with **A**, and white's territorial points are marked with **B**. (The unmarked intersections are neutral points that count toward neither player.) The final score, taking into account captured stones and komi, has white ahead by 4.5 points of territory.

2.2 Computer Go

From a computer scientist's perspective, what stands out about Go is that it remains a perplexing computational problem. The best computer programs for Connect-4, Mancala, Othello, checkers, chess, backgammon, and Scrabble all play at or better than the level of the best human players. Although more time and effort has been spent developing computer programs for Go than for any of these other games except chess, any reasonably skilled club player can defeat the best computer Go programs [8].

Most programs for strategy games, including those for chess and checkers, utilize some variation of brute-force game-tree search: the programs examine all sequences of play up to a certain number of moves and then choose the move that seems to lead to

the most advantageous position. This approach does not work well with Go for two reasons. First, the search tree for Go is exponentially larger than other games. For example, the average branching factor (number of available legal moves) in chess is about 35, whereas the average branching factor in Go is about 250. Second, and more significantly, accurate heuristic evaluation functions for Go are far more computationally expensive than for other games. Unlike chess, where the difference in material is easily computed and provides a reasonable estimate of which side is winning, Go positions have no easily extracted features from which to quickly estimate the game state.

Researchers throughout the world continue to work on computer Go and gather regularly at computer Go tournaments to pit their updated programs against each other [5]. While they continue to make incremental improvements, many believe that a significant breakthrough, perhaps one involving much better recognition of good and bad patterns of stones, is necessary for substantial progress.

3 Using Go in Capstone and Undergraduate Research Projects

Having provided an overview of game play in Go and its characteristics as a computational problem, we now discuss some factors that make it a particularly attractive and suitable problem domain for capstone and undergraduate research projects.

First, and perhaps most significantly, the task of developing a computer Go player is a genuine academic research problem that undergraduate students can readily comprehend. By studying what approaches others have tried, examining the strengths and weaknesses of those efforts, and exploring their own ideas, students gain valuable, hands-on insights about what academic research is and how it is done.

Of course, the expectation of an independent study or undergraduate research project is not for the student to make a significant breakthrough. If structured properly, however, it can provide a sense of the challenge and excitement in pushing at one of the frontiers of human understanding. Furthermore, it can lay the foundation for students to pursue future work in computer Go or a related field.

Second, the task of writing a reasonably complete Go program, even if just to let two humans play against each other, requires students to apply and integrate concepts and techniques from many areas of computer science. For example, writing the core of the control program requires students to design fairly complex data structures that represent the complete state of the board; developing a user interface requires students to design a graphical interface and to implement facilities for saving and loading games and for printing game records and board snapshots; and, providing the ability to play against other players over the internet requires students to understand and implement a networking client.

Of course, adding a computer player will involve exploring a variety of ideas and methods from artificial intelligence. These may include search, knowledge representation, expert systems, pattern recognition, and machine learning.

Third, working on a standard, well-known problem gives students experience interacting with a community of researchers who share a common interest. Students can examine saved games that others have made publicly available, they can test their programs against other programs, and they can ask questions and exchange ideas with others members of the computer-go mailing list [3].

Last, but not least, working on computer Go is fun. Students enjoy playing the game and thinking about ways to teach a computer to do so. For many of the projects described below, students can easily measure their progress (or lack thereof) by playing their programs against each other or against standard, publicly available computer opponents [1].

4 Sample Projects and Commentary

Over the last three years, we supervised eight undergraduate students on Go-related projects in the context of independent study projects, senior theses, or undergraduate research projects. In this section, we discuss a number of these. For each, we describe the actual task and share comments from our own experiences, with the aim of giving instructors a sense of how well these ideas might fit with their students' abilities and educational interests.

4.1 Implement a Go Program for Two Human Players

Description: The most straightforward Go-related project involves developing a program that allows two humans to play against each other. Over the course of one academic year, two of our students worked on this as an independent study project. They designed and implemented data structures and algorithms to determine the legal moves at any point in the game and the board position resulting from making a move. They designed and implemented a graphical user interface that allowed players to start a new game, make moves, undo moves, and save and load games. They also added some of the features mentioned in Section 4.2.

Comments: To a greater degree than the other ideas presented in this section, this project provides students with the opportunity to engage the entire software development process, including design, implementation, testing, and revision. Given the myriad components of a Go program and the relative complexity compared with implementing other strategy games, this project is well-suited for work by a small team of two or three students.

4.2 Add Features to an Existing Go Program

Description: Starting with the program described in Section 4.1, we supervised several projects in which students added features to our existing program. (Source code for a number of publicly available Go programs in varying stages of development can be obtained by following the links from [1].)

In one project, a student added annotation and printing functionality: pieces could be numbered in the order they were played, intersections could be marked or highlighted, and board snapshots and game records could be printed. Figure 3 shows an annotated snapshot from the program. Black's territory is marked with **A**, and white's territory is marked with **B**. Two other projects involved adding the ability to save and load games to disk in **sgf** (Smart Game Format), the standard file format in the computer Go community, and adding the ability to play over the internet against opponents via IGS (the Internet Go Server).

Comments: In general, these projects are less time-consuming than the full implementation project described in Section 4.1. Nevertheless, students gain valuable experience reading and understanding the code for the Go program itself, and they have the opportunity to study a particular topic in substantial depth (e.g., networking protocols for building an IGS client).
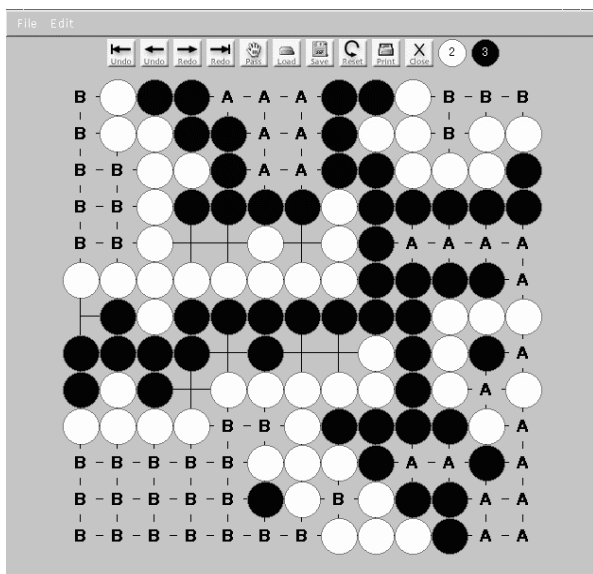
Figure 3: An annotated board position. Black's points are marked with A and white's with B.

4.3 Implement Parts of a "Standard" Computer Player

Description: Most of the best Go programs today use a combination of heuristics, pattern matching, and highly selective search to choose moves to play [2,8]. We have supervised several projects in which students have added to our existing program various components for a computer player based on this "standard" approach.

One project involved identifying certain patterns that occur frequently in Go games. In particular, *ladders* and *nets* describe ways in which stones can be captured. Our student implemented a heuristic selective search mechanism to identify both of these kinds of patterns.

Another project involved automatically determining the score at the end of the game. This requires identifying whether groups of stones surrounded on the outside are *alive* or *dead*. For example, the white stones in the upper right corners of both Figures 1 and 3 are alive, but the white stones in the upper left corner of Figure 1 and the middle right section of Figure 3 are dead. Accurately determining life or death is tricky; even very strong players sometimes disagree over the life or death status of a group of stones!

This project is especially valuable because the computations involved in computing the score at the end of a game are closely related to the computations involved in estimating the score during the game. Thus, this can assist with efforts to develop a general heuristic evaluation function for Go positions.

Comments: Projects based on implementing components of a standard computer player are well-suited for students who wish to learn about the state of the art in computer Go and who wish to work directly on a computer player. By trying out their own ideas and comparing their results with others, students learn about academic research in computer science, particularly artificial intelligence.

It is important, of course, to give students realistic goals and expectations. We have found that projects based on implementing components of a "standard" computer player are challenging but

feasible. They provide realistic stepping stones along the way toward the much more difficult task of implementing a full computer player.

4.4 Implement a Computer Player that Learns

Description: For more advanced students, Go provides a wonderful opportunity to explore novel artificial intelligence approaches to developing a computer player. We have worked with two students on using reinforcement learning methods to select moves. Starting with the source code for GNUGo, a relatively strong publicly available Go program [4], our students designed and implemented a neural network-based evaluation function whose weights were learned using temporal difference learning [9], a type of reinforcement learning that worked well for the TD-Gammon backgammon program [10]. In this approach, the program repeatedly played games against itself and updated the weights based on the outcome of each game.

The computational requirements of this technique forced our students to stick with small board sizes: 5 x 5, 7 x 7, and 9 x 9. Nevertheless, they produced encouraging results indicating that the system was learning reasonable concepts. For example, Figure 4 illustrates what the program learned about the best first move of the game after playing about 4000 games on a 7 x 7 board against itself. It estimated that playing in the center was the best move, playing in one of the diamond-marked intersections was next best, and playing in one of the **X**-marked intersections was next best after that. While the estimates are not perfect, they represent a substantial improvement over simply treating each intersection as equally good.
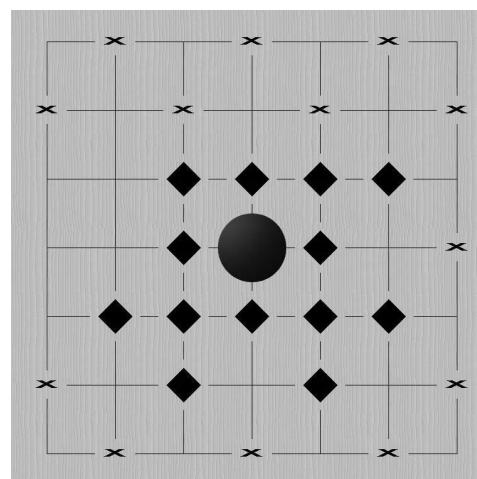
Figure 4: By playing multiple games against itself, a program based on temporal difference learning identified the center intersection as the best place to start a game, followed by the diamond-marked intersections, followed by the X-marked intersections.

Comments: Like those in Section 4.2, this project also gives students the opportunity to learn about the state of art in computer Go. Since it involves a novel approach to constructing a Go player, the risks and rewards are higher. Given realistic expectations, guidance regarding which ideas to pursue, and incremental, measurable goals, our students enjoyed the

opportunity to simultaneously learn about advanced topics not covered in the classroom and work on the implementation of a computer Go player.

5 Conclusion

In this paper, we have provided an overview of the game of Go and the characteristics that make it a unique and challenging problem, one that is well-suited for a variety of capstone and undergraduate research projects. Furthermore, we have described our experiences advising students on numerous computer Go projects over the last three years. This appears to be the first comprehensive treatment of how computer Go can be used for advanced undergraduate projects in computer science.

Overall, we are pleased and satisfied by the way in which the projects have encouraged and motivated students to integrate what they have learned about different areas of computer science, to independently investigate advanced topics, and to design, implement, and experiment with their own Go programs. With projects focused on specific components of a computer Go player or on somewhat simpler problems (e.g., games played on smaller board sizes), Go is an ideal environment in which students can learn about and pursue academic research.

While some computer science educators may not have much familiarity with the game of Go, we believe that for many of them, the time spent learning the basic rules and computational challenges will translate into a variety of meaningful and rewarding opportunities for students.

6 Acknowledgements

References

[1] British Go Association. BGA Software Catalogue, 2002. Online. Internet. Available WWW: http://www.britgo.org/gopcres/gopcres1.html

[2] Burmeister, J. and Wiles, J. AI Techniques Used in Computer Go. In *Proceedings of the Fourth Conference of the Australasian AI Society* (1997).

[3] Computer Go mailing list. Online. Internet. Available email:computer-go@lists.uoregon.edu with message body reading "subscribe computer-go".

[4] GNU Go Project. GNU Go Homepage, 2002. Online. Internet. Available WWW: http://www.gnu.org/software/gnugo/gnugo.html

[5] Hafner, K. In an Ancient Game, Computing's Future, 2002. New York Times Online. Internet. [August 1, 2002]. Available (registration required) WWW: http://www.nytimes.com/2002/08/01/technology/circuits/01GONE.html

[6] Iwamoto, K. *Go for Beginners* (1972), Ishi Press.

[7] Kim, J. and Soo-hyun, J. *Learn to Play Go, Volume 1, 2nd Edition* (1997), Good Move Press.

[8] Müller, M. Computer Go. *Artificial Intelligence Journal*, 134:1-2 (2002), Elsevier, 145-179.

[9] Schraudolph, N., Dayan, P., and Sejnowski, T. Temporal Difference Learning of Position Evaluation in the Game of Go. In *Advances in Neural Information Processing Systems* 6 (1993), 817-824.

[10] Tesauro, G. Temporal Difference Learning and TD-Gammon. In *Communications of the ACM* 38:3 (1995), 58-68.