Ponnuki, FiveStones and GoloisStrasbourg: three software to help Go teachers

Tristan Cazenave

Labo IA, Université Paris 8, 2 rue de la Liberté, 93526, St-Denis, France
cazenave@ai.univ-paris8.fr

Abstract. This paper describes three software that are used to teach beginners the game of Go. Ponnuki plays the game of ponnuki-go which consists in capturing first a stone on a small board. It can be used with different sizes for the board and different configurations of play. FiveStones plays the game of capturing five stones, it is an intermediate game between ponnuki-go and the real game of Go. GoloisStrasbourg plays the game of Go on a 9x9 board. It counts the territory with the strasbourgeoise rule which simply consists in counting the stones present on the board at the end of the game. This rule is very easy to understand even for beginners, and GoloisStrasbourg enables beginners to learn Go by themselves, possibly without assistance. For the three software, the paper describes the methods used to implement them and some problems and solutions encountered in using them in practice.

1 Introduction

In this paper, we present three programs that can be used to help teach the game of Go. These three program are designed for teachers that use the teaching method consisting first in teaching Ponnuki-Go, then in teaching how to capture five stones, and eventually playing Go, counting the number of stones of each player on the board at the end to determine the winner [1].

The first section details computer programs for the game of Ponnuki-Go. The second section is about programming the game of capturing five stones, and the third section about GoloisStrasbourg, a program that plays Go according to the strasbourgeoise rules.

2 Ponnuki

Ponnuki-Go consists in capturing first a stone of the opponent. It is easy to teach as the basic aptitude needed to play the game is to count liberties. Some programs have been written to solve the game on small boards. We will describe them in the first subsection. The second subsection is about our 9x9 Ponnuki-Go program.

2.1 Solving 6x6 Ponnuki-Go

I solved 6x6 Ponnuki-Go with a cross cut in the center in 2002 [2], and Erik van der Werf [3] solved the version with an empty board in 2002 too.

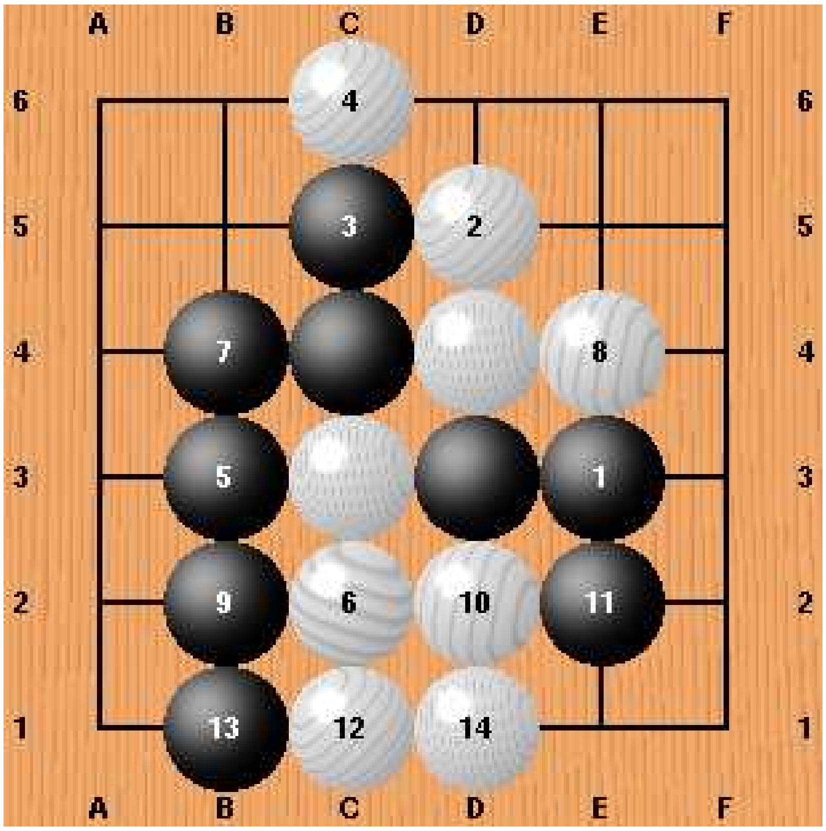

Fig. 1. The solution to 6x6 Ponnuki-Go with a cross-cut found by GTS(6,3,2,0).

Our program is based on an optimized Alpha-Beta. The optimizations include the use of transposition tables, containing the score and the best move, the memorization and use of two killer moves after the transposition move, the history heuristic with a weight of 2^{Depth}, and an incremental evaluation function which computes the difference between the number of liberties of the black string that has the least liberties and the number of liberties of the white string that has the least liberties. The number of liberties of strings are updated incrementally too. These optimizations are similar to the optimizations used in [3] to solve 6x6 Ponnuki-Go with Alpha-Beta.

Using a new algorithm based on generalized threats [4], we were able to reduce the time needed to solve 6x6 Ponnuki-Go with a cross-cut in the center. The solution found by this algorithme is given in the figure 1.

While debugging our program, we found interesting positions, that need some subtle play. An example is given in the figure 2.

2.2 Playing 9x9 Ponnuki-Go

My Ponnuki-Go program is based on an Alpha-Beta algorithm. The evaluation function is the same as for the 6x6 version: the difference between the number of liberties of the computer string that has the least liberties, and the number of liberties of the opponent string that has the least liberties.

The optimizations of the Alpha-Beta are the same as the optimizations used for the 6x6 version.

3 FiveStones

FiveStones is my program that plays the game of capturing five stones. The algorithm used to play is an optimized Alpha-Beta. The evaluation function consists in computing the difference between the number of stones captured by the computer and the number of stones captured by the opponent. The resulting number is multiplied by 100, and the difference between the number of liberties of the computer string that has the least liberties, and the number of liberties of the opponent string that has the least liberties is added to the evaluation.

4 GoloisStrasbourg

4.1 Golois

Golois is a search based Go program. It uses an optimized Alpha-Beta and the Generalized Threats Search algorithm [4] to solve tactical problems.

In this section, we detail the architecture of Golois. We start with defining the possible subgoals that can be used in the tactical part of

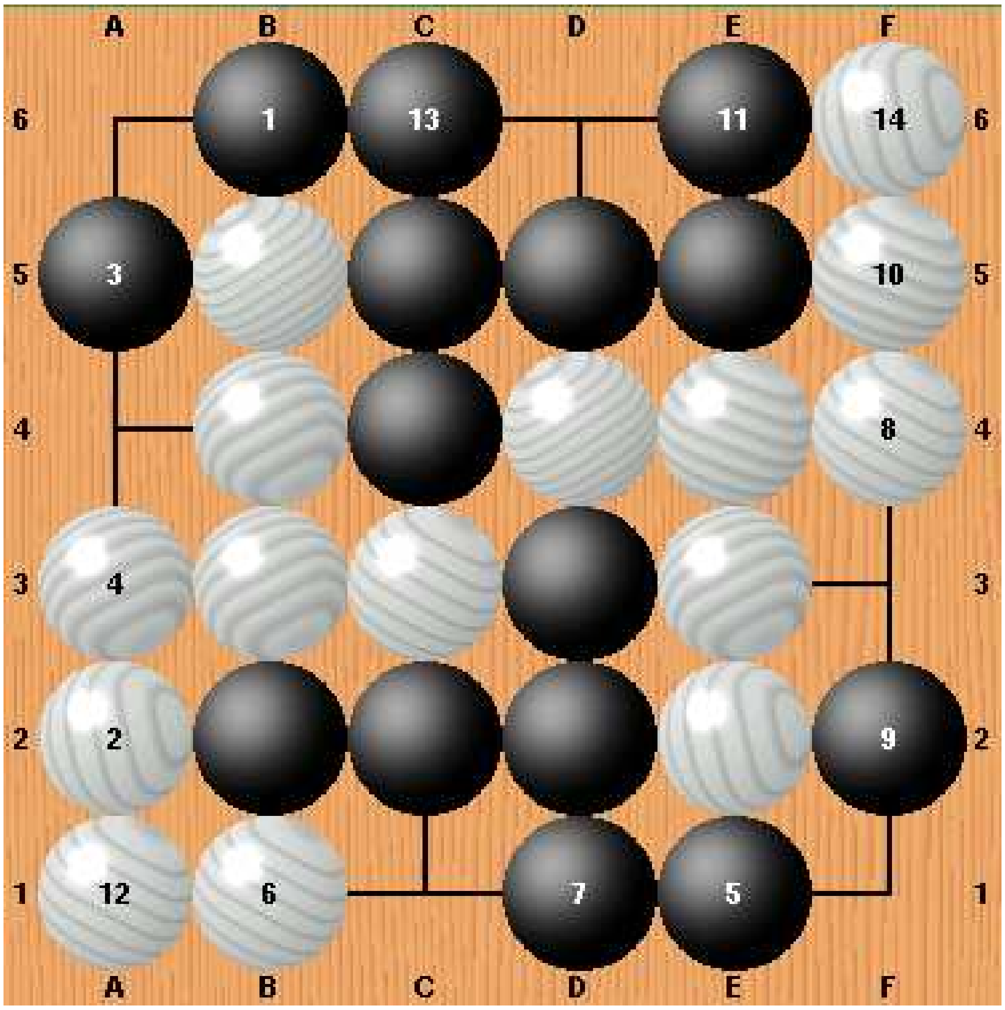

Fig. 2. An amusing 6x6 Ponnuki-Go problem.

Golois. Then we define the possible states resulting from tactical search. We then show how groups are built using search results on connections.

4.2 Subgoals of the game of Go

In order to evaluate a position and choose a move, a Go program has to solve many sub-problems. Examples of subgoals that can be solved are:

- capturing a string / saving a string.
- connecting two strings / disconnecting two strings.
- making an eye / killing an eye
- making a group live / killing a group
- deciding the status of a semeai

4.3 Possible states for search results

The evaluation of subgames in the game of Go returns integer values. Some values are special like the Won and Lost values that are extremal. They are next to the Won by ko and Lost by ko values, which are not always terminal values but that can be terminal if no other option is better (if a variation finds Won by ko, there is still some hope that another move leads to a Won value). All other values strictly greater that Lost by ko, and strictly lower than Won by ko account for an Unknown result. An Unknown result means that further search can transform it to one of the four terminal values. In the following, we will only consider three different values as possible results for a tactical search: Lost, Unknown and Won.

Most Go problems are associated with two search results. One result is associated with the friend player playing first, and the other with the enemy player playing first. This representation has links with the decomposition approach of the game of Go and is used by most Go programs. It is much more efficient than brute force search when problems are independent [5, 6].

We use a notation derived from Conway's theory. A result is noted with a left and a right part, separated by a '$|$' and enclosed in braces. For example the status of a string that can be captured if the friend player (Left) plays first, and which status is unknown if the opponent player (Right) plays first is noted $\{Won|Unknown\}$.

A Won or an $Unknown$ result for Left is associated to a set of moves. A $Lost$ or an $Unknown$ result for Right is associated to a set of moves.

$Lost$ results for Left and Won results for Right are not associated to any move because Left is aiming at finding winning moves and Right is aiming at finding moves that make Left lose.

Each game is associated to a color, which is the color of the Left player. We define the function $Color(G)$ that returns the color of the game G. A game is also composed of a $Goal$, a $Threat$ and a $Result$. We define $Goal(G)$ the goal associated to the game G, $Threat(G)$ the next threat to be tried for solving the game G, and $Result(G)$ the result of the game which is noted $\{R_1|R_2\}$ with R_1, R_2 in $\{Lost, Unknown, Won\}$.

4.4 Tactic, groups and strategy

The high level reasoning of a Go program reasons on groups. Groups are sets of connected strings. They have properties such as the number of eyes, the enclosed territory, the influence, the list of friend groups they can connect to, or the list of captured strings among other properties. The evaluation of the strength of a group is performed using these properties. Groups are built according to the results of search on connections. Some of the properties such as the list of captured strings, the eyes of the group, or the life and death status are computed with dedicated search algorithms.

In order to build groups, many tactical search have to be performed. It is therefore usual in Go programs to separate reasoning in two phases: the tactical phase and the strategic phase. The tactical phase computes captures, connections, eyes and life and death. The strategic phase builds the groups according to the tactical results and then evaluates the position and chooses the relevant global moves [7, 8].

4.5 Features of a group

A key component of a Go program is the evaluation of the safety of groups. The evaluation of the safety of a group is based on many properties. In this subsection, we give the properties computed for each group in Golois:

- Value: the number of points in chinese rule the group makes if it is alive with its current size, this takes into account the stones of the group and the territory associated to the group.
- nth Liberties: the first order liberties are the union of the liberties of all the strings of the group. The second order liberties are the union

of all the liberties of liberties, excluding the first order liberties. The third order liberties are the liberties of second order liberties which are not first nor second order liberties.
- Life: the group can either be alive, unsettled, dead, or have no life property. The life property can be determined statically or by search.
- Capture: the group can be captured or capturable, this property is only used for groups composed of only one string.
- Enclosed: A group can either be enclosed, enclosable, escapable or escaped. These poperties are mainly based on the number of second and third order liberties.
- Semeai: A group might be in semeai against another neighboring group.
- Neighbors : The group can have neighboring groups, it can connect to if they are friend, or that it can attack if they are opponent's groups.
- Influence: For each stone which is nor dead on the board, an influence is irradiated, the influence of a group consists in the empty intersections neighboring the group that are closer to a computer stone than any opponent stone.
- Territory: the territory is the set of empty intersections neighboring the group which have a shortest path to the group two steps lower than their shortest path to any opponent stone. For example, the second order liberties which have a shortest path to any opponent stone stritly greater than three.
- Prisonners: the prisonners are the string neighboring the group that are captured.

4.6 Evaluation of groups safety

The evaluation of groups safety is a multi-step process. There are two evaluation functions for a group. The first one is a rough evaluation that roughly evaluates if the group is alive, dead or in between. It consists in:

- computing the value of the group, as the maximum of the influence and of the territory, plus the number of stones, plus the number of prisonners.
- then to evaluate statically the life of the group
- then to set the strength of the group at one if the life evaluation is Won, or if the influence is greater than twenty, or if the territory is greater than twelve, or if the number of prisonners if greater than eight.

– otherwise to set the strength at zero if the group is enclosed, and loses the semeai to all its neighbors, or if the group is captured.

Once the properties and the rough evaluation function are computed for each group, Golois joins groups that are neighboring dead groups. For each of these groups it then computes an elaborate evaluation function that takes into account the strength of neighboring groups. The process of joining groups around dead groups, and of evaluating the groups with an elaborate evaluation function is iterated until no new dead groups are found. This iteration at the global level in order to stabilize the evaluation function is similar to the behavior of the strategic level of Indigo [9].

The elaborate evaluation function computes all the features of the rough one, and also:

– it sets the strength to zero if the group is either captured, evaluated as dead or is enclosed with less than three influenced intersections.
– the group is considered hopeless when the maximum strength of neighboring friend groups is zero, the minimum strength of opponent neighboring group is one, the influence size is less than five, it is enclosed and cannot live.
– the strength is set to 0.5 if it can live or live by ko, or if the maximum strength of neighboring friend groups is one, or if it is possible to kill a neighboring opponent group, or if it is escaped (enough second an third order liberties), or if the influence size is greater than 6.

Performing a search to find if a group is alive or not, or if it can win a semeai or not is expensive in CPU time. Therefore, this search is only performed for enclosed group that have a strength 0.5. It eliminates many useless search for groups that are clearly strong, but that may require a deep search to make two eyes.

4.7 Evaluating influence

The irradiation of the influence of groups is based on the shortest path from an empty intersection to the closer group which is not dead. For all the non dead groups on the board, the liberties of the group are noted as influenced by the group. If an intersection is a liberty of two groups of opposite color, it is not counted as influenced by any color. For all the influenced empty intersections, and for the two colors, all the empty neighbors which have not already been seen in the process are marked as influenced. Again an empty intersection has two shortest pathes of equal length to two groups of opposite colors, it is marked as not influenced. This process is repeated seven times. In the end it gives a reasonable evaluation of influence, even in the case of relatively large moyos. Especially in this case of large moyos, it gives better results than the traditional way of computing influence as an exponentially decreasing function.

4.8 Evaluating global moves

Choosing moves at the global level is currently performed using an approximation of the temperature of the moves. For each subgame in $\{Won|Lost\}$ or $\{Won|Unknown\}$ or $\{Unknown|Lost\}$, two sets of moves are associated, one for the friend player and one for the opponent player. Each of these moves is played and the position is re-evaluated after each move. Therefore, each move is associated to an evaluation that approximates the difference in territory the moves makes. The final value of a friend move is an approximation of its temperature. The temperature is approximated by substracting the value of the opponent moves that are prevented by the friend move to the difference in territory the friend moves makes.

I am currently investigating the use of threats values [10], and the use of a global quiescence search for evaluating global moves.

4.9 Playing after the endgame is over

In order to play according to the Strasbourgeoise rule, some modifications to my playing engine have been made. Under the japanese and the chinese rules, Golois stops playing as soon as no move has a value strictly above zero. The scoring of moves is based on the chinese way of counting. Therefore, moves on the neutral points, at the end of the game, have a value of one point. I have modified the way the moves are played at the end of the game in order to play according to the Strasbourgeoise rule.

A move can be played if its value is strictly greater than zero, or if the intersection does not belong to the opponent. The move with the highest value is selected. In case two moves have the same value, the program counts the number of opponent stones neighboring the intersection of each move, as well as the number of empty neighbor intersections. It chooses in priority the move that has the most neighboring opponent stones. In case of equality, it chooses the move that has the most empty neighbors.

5 Conclusion

I have described Ponnuki, FiveStone and GoloisStrasbourg. These three software where written with the hope they will help Go teachers, and also that they can be used by people to teach themselves to play Go alone.

References

1. Fenech, A.: Le Go un jeu d'enfant. Chiron (2003)
2. Cazenave, T.: La recherche abstraite graduelle de preuves. In: Proceedings of RFIA-02, Angers, France (2002) 615–623
3. van der Werf, E., Uiterwijk, J., van den Herik, H.: Solving ponnuki-go on small boards. In Uiterwijk, J., ed.: The 7th Computer Olympiad Computer-Games Workshop Proceedings, Maastricht, The Netherlands, IKAT, Department of Computer Science, Universiteit Maastricht (2002) 5–11
4. Cazenave, T.: A Generalized Threats Search Algorithm. In: Computers and Games 2002. Lecture Notes in Computer Science, Edmonton, Alberta, Canada, Springer (2002)
5. Conway, J.H.: On Numbers and Games. Academic Press, London/New-York (1976)
6. Mueller, M.: Decomposition search: A combonatorial approach to game tree search, with applications to solving go endgames. In Dean, T., ed.: IJCAI 99. Morgan Kaufman, Stockholm, Sweden (1999) 578–583
7. Bouzy, B., Cazenave, T.: Computer Go: An AI-Oriented Survey. Artificial Intelligence **132** (2001) 39–103
8. Mueller, M.: Computer go. Artificial Intelligence **134** (2002) 145–179
9. Bouzy, B.: Modélisation cognitive du joueur de go. Phd thesis, Université Paris 6 (1995)
10. Cazenave, T.: Comparative evaluation of strategies based on the value of direct threats. In: Board Games in Academia V, Barcelona, Spain (2002)