# The Go-Playing Program Called Go81

Tapani Raiko

Helsinki University of Technology, Neural Networks Research Centre
P.O.Box 5400, FIN-02015 HUT, Espoo, Finland

**Abstract.** Go is an ancient game, for which it has proven to be very difficult to create an artificial player. Go81 is yet another try in that direction. The main idea is as follows: firstly, create a so called ant that tries to play as well as possible, given that it has to be very fast and slightly randomized. Secondly, use these ants to play the game from the current state to the end several times and make use of the information from these possible futures. This approach avoids the evaluation of an unfinished game, which is perhaps the one thing that makes computer Go so difficult. Two versions of Go81, one for Palm and one for a Linux console, are tested against a shareware program AIGO for Palm and an open source project GNU Go accordingly. The Palm version is as strong as AIGO and the console version is two stones weaker than GNU Go on a 9 by 9 board. The proposed approach can also be used to generate interesting data to be studied with machine learning techniques.

## 1 Introduction

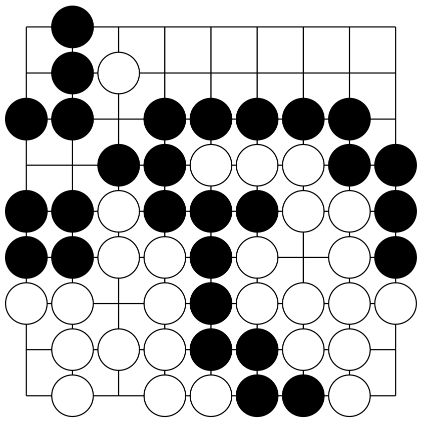
The introduction describes the game of Go and the current state of computer programs that play Go.

### 1.1 Go

"Go is an ancient game that originated in China, with a definite history of over 3000 years, although there are historians who say that the game was invented more than 4000 years ago. In this game, each player tries to exert more influence on territory than her opponent, using threats of death, capture, or isolation. It is, therefore, a symbolic representation of the relationships between nations. Go is getting increasingly popular around the world, especially in Asian, European and American countries, with many worldwide competitions being held." (From http://senseis.xmp.net/)

Two players, black and white, alternately place stones on the empty points of the board. Players may also pass. The standard board is 19 by 19 (i.e. the board has 19 lines by 19 lines), but 13-by-13 and 9-by-9 boards can also be used. The game starts with an empty board and ends when it is divided into black and white areas. The one who has a larger area wins. (See Figure 1)

Stones of one colour form a block when they are 4-connected. Empty points that are 4-connected to a block are called its liberties. When a block loses its last



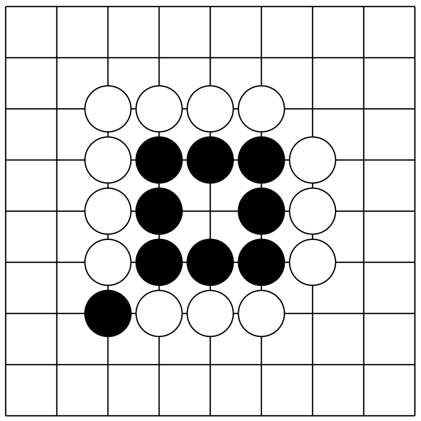


**Fig. 1.** The rules of Go. Left: The end of a game. Black has won, because he controls a greater area on the board. The lone white stone in the top is assumed dead. Right: White has three blocks and black two. White can capture the block of 8 black stones by playing in the middle and thus removing their last liberty. Captured stones are removed from the board.

liberty, it is removed from the board. (See Figure 1) After each move, surrounded opponent blocks are removed and only after that, it is checked whether the block of the played move has liberties or not.

There are different rulesets that define more carefully what a "larger area" is, whether suicide is legal or not, and how infinite repetitions are forbidden. For a more detailed description with examples, see
http://www.britgo.org/intro/intro2.html

## 1.2 Computer Go

"Of all games of skill, Go is second only to chess in terms of research and programming efforts spent. Yet in playing strength, Go programs lag far behind their counterparts in almost any other game. While Go programs have advanced considerably in the last 10-15 years, they can still be beaten easily by human players of moderate skill." [14]

Static board evaluation in Go is very difficult. This combined with a large branching factor makes it sure that regular negamax-type algorithms won't work very well. One of the reasons behind this difficulty is the fact that there are stones on board that will eventually be captured, but not in near future. In many cases experienced Go players can classify these dead stones with ease, but algorithmically, the problem is polynomial-space hard [7]. Using a simple look-ahead to determine the status of stones is not always easy since it might take dozens of moves to actually capture the stones.

One of the reasons that Go is thought to be difficult for computers is simply the fact that (many) humans are amazingly good at Go. The visual nature of

Go fits human perception but is hard to model in a program. Computers have difficulties sorting out pictures of chairs from pictures of bicycles - something which is quite trivial for humans. Consequently, humans are able to recognize subtle differences in Go positions. They can judge very early on, whether a large, loose group of stones can be captured or not. Humans can answer such questions with just a glance on the board, whereas an equivalent proof by a computer search seems completely out of reach. [14]

Almost all Go programs contain a pattern database and a pattern matching subsystem [14]. These patterns try to mimic the visual thinking that humans are supposed to do. A large effort is done to handcode, to test and debug these patterns. There is also some research on automatically acquiring patterns [10] from game collections.

Local searches are used to answer questions like "can these stones be connected?" or "can this group survive?". There are methods like lambda-search [16] tailored for such problems. It is still an open question, how to combine all the information gathered from these search trees and use it globally. The search tree contains much more information than the yes/no answer, like threaths etc.

## 2 Overview of Go81

Go81 is a Go-playing program for the Palm handheld computers. This hardware has some limitations, but the program can cope with them. The current version 1.6 is less than 40 kilobytes in size and works reasonably fast even with a 20 MHz processor. It is mainly designed for playing 9 by 9 games (hence the name Go81) but can also manage 13 by 13. The program is available with the source codes [15].

Go81 avoids board evaluation during the game by using "ants". Ants are simple Go-playing agents that evaluate moves directly using only local information. This makes them fast to use, since it is easy to update only those move evaluations that are close to the latest move. The idea is to play the game to the end several times from the current situation by using a swarm of ants. The information gathered from the end states is then used to select the next move.

### 2.1 Ants

An ant evaluates every legal move by looking at the following local information: 1) The 3-by-3 neighbourhood of the point; and 2) the liberties of the blocks whose liberty count would be affected. Many fundamentals of the game of Go appear already in such a small neighbourhood as the 3 by 3. These include cuts, connections, walking stones, empty triangles, and (simple) eyes [13] (see Figure 2). Liberties are important in fighting, since they define when blocks are captured. Evaluations are defined here such that the burden of a block to its owner is directly relational to its size and diminishes exponentially when the block has more liberties. This makes sense, since it is more important whether a block has 1 or 2 liberties, than whether it has 11 or 12 liberties. The move evaluation has also some random noise added to it.

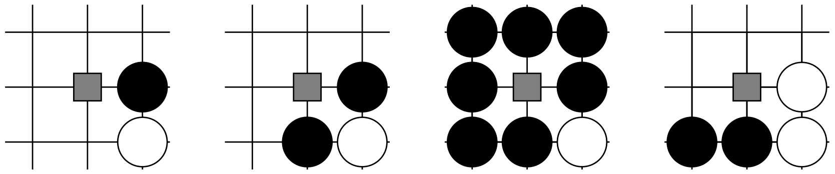


**Fig. 2.** Examples of 3-by-3 neighbourhoods from left to right: 1. Stones go walking. A good place to play. 2. Connection for black or a cut for white. Very good for either player. 3. Simple eye. Black must avoid playing in there. 4. Empty triangle for black so black should avoid playing there.

## 2.2 Swarm intelligence

Real ant colonies appear to be very intelligent in some situations. For instance, every ant seems to know the shortest way to the food source. Yet this intelligence rises from very simple interaction among individuals (following the trail left by others). Swarm intelligence[4] has been applied in computer science to problems like rerouting traffic in a busy telecom network. The idea is that simple problem solvers are used in swarms and parts of those answers that happen to be successful, are reused and combined by others later on. Ants are typically stochastic, so that they cover the search space better.

In Go81, the ants are used to play the game twice to the end, once starting as black and once as white (see Figure 3). The game can be played further than people usually do, really capturing all the dead stones and filling own areas. This way, it is trivial to determine which areas belong to black and which to white. The areas, where these two possible futures differ, are the interesting ones. It is not yet determined, to whom they belong to. The blocks that might surround others or be surrounded and the border between black and white territories are the places where the focus of the game is. The swarm of two ant battles is used thus to rule out uninteresting areas. Also, it seems that large continuous neutral territories are more interesting than small ones.

The possible futures are used also to find vital points. If there is a group of stones in the end that barely survived (it has no more than three eyes), the points that form its eyes are marked important. When a particular point is important to many eyes at once, it is called vital. The evaluation of playing in the vital point is increased for both players. Figure 3 shows an example of a vital point.

To conclude the evaluation of moves in Go81, the different terms are listed here: Evaluation = randomness + 3-by-3 shape + liberty changes + area interest + vitality. Note that moves are evaluated as opposed to evaluating the position that the move leads to. Only the liberty change term is defined such that it could be seen as the difference of the evaluations of the game state before and after the move.

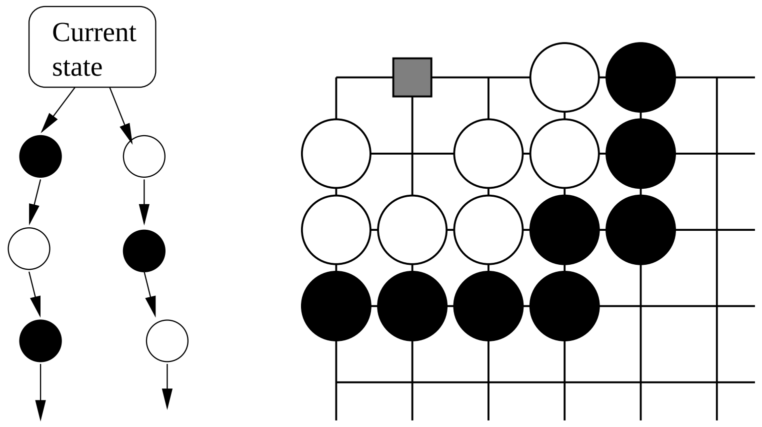


**Fig. 3.** Left: The search tree of Go81. The swarm consists of only two ants. Right: The vital point for white. If white is the first to play there, he forms three eyes (the empty points around the vital point, and the stones survive. If black plays there first, he will capture the whole corner.

## 2.3 Benson's algorithm

Benson's algorithm [3] can be used to determine which stones can never be captured even if the opponent would get many moves in a row. The algorithm is ran only for the current position and the result is used for three purposes: 1) The player never plays in an area that can be proven safe; 2) Blocks that are proven safe are no burden at all, when ants evaluate liberty changes (see Section 2.1); 3) There can be no vital points (see Section 2.2) in an area that is proven safe already. In practice, the game might end before any stones can be proven safe, so Benson's algorithm is not always very useful.

## 3 Related work and Go81Console

The idea of playing the game stochastically to the end many times from the current position originates from Abramson [2]. He suggested that the expected outcome of random games provides a heuristic evaluator that is not only domain-independent and easy to compute but also accurate at least in some domains. Results were good in Tic-tac-toe and Othello and he also experimented in Chess. The idea was applied to Go by Bruegmann [6] and recently developed further by Bouzy et al. [5]. Their idea is quite comparable to Go81. Both avoid evaluating unfinished game states. Their ants are even more naïve – they only avoid playing inside one's eyes. Otherwise the move evaluation is given by the swarm AI in the form of giving a value (a smell trace, if you wish) to each point on the board, regardless of when in the future the point will be used. These values are changed based on the outcome of these random games by trying to estimate the expected outcome of the games where the particular point was played. As one of the perspectives of their work, they mention the possibility of using patterns for the ants, which would bring the ants quite close to Go81.

Go81Console combines Go81 with smell traces. It is assumed that the smell trace becomes more inaccurate when the game progresses, so the ant follows the smell trace more closely in the near future and becomes more independent towards the end. Similarly, a larger weight is given to the moves played in the near future, when adjusting the evaluations. The program plays the game 100 times to the end from the current situation before selecting the next move (simply taking the largest evaluation in the gathered table). It is thus unreasonably slow for current handheld computers but quite fast for a desktop computer.

## 4 Experiments

Go81 is played against a shareware program AIGO [12] version 2.0.0 for Palm. AIGO plays quite fast and is about 170 kilobytes in size. It seems to be the only other program that plays reasonable Go on the Palm handhelds. Both programs won 5 out of 10 games and both made large and clear mistakes during the games. Go81 won with a larger margin on average, which might indicate that it is stronger in fighting but weaker in territory oriented style.

Go81Console is tested on a 9-by-9 board by letting it play against GNU Go [9] version 3.4, an open source project that started on 1989 with at least 29 people involved so far. GNU Go won the Go competition at the Computer Olympiad 2003 winning all of its 10 games. Go81Console was given a 2 stone handicap, which means that it can start with two stones already on the board. This seems to be enough, since Go81Console won 6 and lost 3 of the 10 games (with one draw). GNU Go is much stronger in reading out complicated fights. The games that it won were in fact big fights, where Go81Console lost many of its stones. Without the 2 stone handicap, Go81Console was crushed most of the time, but actually managed to win once by one point.

Finally, Go81 is tested against Go81Console. One would expect that Go81Console is stronger, since it is 50 times slower than Go81 (but still much faster than GNU Go). This proved to be true: Go81Console could give three handicap stones to Go81 to balance the strengths. The console version still won 6 out of 10 games. The relative strengths of the programs are summarized in Figure 4.

## 5 Discussion and future work

The swarm intelligence tries to avoid the two main problems in building an artificial player for Go: large branching factor and difficult evaluation of game positions. The exponential growth of the game tree is avoided by exploring a constant number of nodes at each depth. Difficult evaluation is avoided by evaluating positions only in the very end of the game when evaluation is trivial.

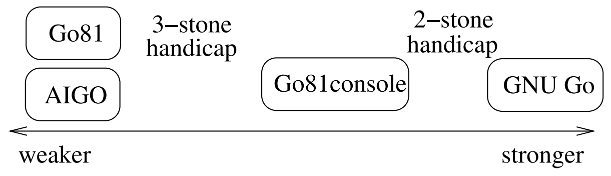


**Fig. 4.** The relative strengths of some Go-playing programs on 9-by-9 board.

Swarm intelligence can also benefit a lot from parallel processing, since different ants can be separated to different processes easily.

There are numerous ways to improve Go81: One could tune numerical parameters in the program by using for instance genetic algorithms [1]. The ants seem to get distracted when there are dead stones on the board. They waste moves to actually capture them, when one could just wait and do more important moves elsewhere. The ideas of Bouzy et al.[5] seem to help in that. To make Go81 more fit to play on bigger boards, one should consider patterns larger than 3 by 3. They would be especially usefull in the opening, so perhaps the patterns with just a few stones would suffice. Also, the swarm provides an estimate on who is leading the game. When ahead, one should be more defencive and vice versa.

One interesting perspective would be to provide more information to the ants. Especially local search trees answering some tactical problems would be useful. A simple special case of that would be to provide the ants with miai points, that is, if the opponent plays to one of two points, the ant should immediately answer by playing to the other one. This would help to achieve a better level of reading complicated tactical fights.

Go81 plays the game only twice to the end to determine the focus of the game in sense of large undetermined territories. This can be generalized in case more information than just the two games are available. A promising way to do that seems to be the covariance between the outcome of a certain point with the score (i.e. the sum of outcomes of all points). That is, whenever one of the players thrives in some part of the board, how important it is for the whole game. Figure 5 shows an example situation. Human players tend to agree that the critical area seems to be the one suggested by this heuristic. One could guide the ants in various ways depending on how far they are in the search tree. First they would be guided by expected outcomes (smell traces), then by covariances, and finally, deep in the search tree, letting them function without guidance. Stern et al. [8] model the territorial outcome given the current position by using data from professional games. Perhaps they would benefit from having the covariances in the data in addition to the expectations.

Hyötyniemi searched for patterns in Chess position [11] even if Chess programs are typically not pattern-based. Go-playing programs, on the other hand, quite often use patterns. Automatic acquisition of patterns from Go game records has been studied as well [10]. These ideas can be extended by supplementing the

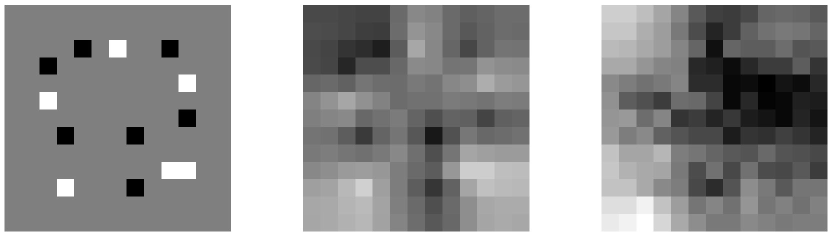


**Fig. 5.** *Left:* An example board position on a 13-by-13 board. *Middle:* Expected outcome for each point. Darker colour represents a greater probability to end up as black players territory. *Right:* Covariance of outcome of each point with the score. Darker colour represents greater covariance.

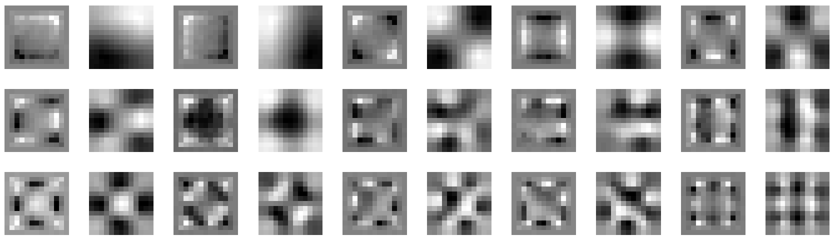


**Fig. 6.** First 15 principal components for data containing the board positions and the expected outcomes on a 13-by-13 board.

board position with other information such as the expected outcome for each point. That way one could also predict the score based on the game state and a learned model. Figure 6 shows a very preliminary result on extracting patterns. 10000 board positions from 13-by-13 games are supplemented with the expected outcome information, and principal component analysis (PCA) is applied to the data. The resulting features resemble closely the features from applying PCA to natural image patches. No wonder it is often said that Go is a visual game.

## References

1. S. Ragab A. Abdelbar and S. Mitri. Co-evolutionary particle swarm optimization applied to the 7x7 Seega game. In *Proc. of the IntJ̇. Conf. on Neural Networks*, pages 243–248, Budapest, Hungary, July 2004.
2. B. Abramson. Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):182–193, 1990.
3. D. Benson. Life in the game of Go. *Information Sciences*, 10:17–29, 1976.
4. E. Bonabeau and G. Thraulaz. Swarm smarts. *Scientific American*, pages 72–79, March 2000.
5. B. Bouzy and B. Helmstetter. Developments on Monte Carlo Go. *Advances in Computer Games 10*, 2003.

6. B. Brügmann. Monte Carlo Go. Technical report, Syracuse University, March 1993. ftp://ftp.cse.cuhk.edu.hk/pub/neuro/GO/mcgo.tex.
7. M. Sipser D. Lichtenstein. Go is polynomial-space hard. *Journal ACM*, 27(2):393–401, 1980.
8. T. Graepel D. Stern and D. MacKay. Modelling uncertainty in the game of Go. In *Proc. of the Conference on Neural Information Processing Systems*, Vancouver, Canada, December 2004. submitted.
9. D. Bump et al. GNU Go home page, 2004. http://www.gnu.org/software/gnugo/devel.html.
10. A. Huima. Unsupervised learning of go patterns. http://people.ssh.fi/huima/compgo/, 1999.
11. H. Hyötyniemi and P. Saariluoma. *Chess - Beyond the Rules.* Finnish Artificial Intelligence Society, 1999.
12. A. Iizuka. AIGO home page, 2004. http://www001.upp.so-net.ne.jp/iizuka/AIGO/.
13. T. Kageyama. *Lessons in the Fundamentals of Go.* Kiseido publishing company, 1978.
14. M. Müller. Computer Go. *Special issue on games of Artificial Intelligence Journal*, 2001.
15. T. Raiko. Go81 home page, 2004. http://www.cis.hut.fi/praiko/go81/.
16. T. Thomsen. Lambda-search in game trees – with application to Go. *Computers and Games 2000, Lecture Notes in Computer Science*, 2001.