Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences xxx (2004) xxx–xxx
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Maximizing the chance of winning in searching Go game trees

Keh-Hsun Chen

Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223, USA

Accepted 26 April 2004

Abstract

The current global selective search and decomposition search in Go typically back up territory scores. This approach is inherently flawed. We propose a new strategy of backing up "chance of winning". We show how an evaluation function on the chance of winning can be constructed. Also we develop a probabilistic combinatorial game model and an algorithm for decomposition search to work with probabilistic outcomes in maximizing the chance of winning.
© 2004 Elsevier Inc. All rights reserved.

1. Introduction

Due to the difficulty of its positional understanding by a machine and its high branching factor, Go is generally regarded as a most challenging game to program.

Global selective search [3–5] is an effective approach to deal with the intrinsic difficulties of Go. Many Go programs use this strategy for move selection

E-mail address: chen@uncc.edu.

0020-0255/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2004.04.009

decisions [7]. The evaluation functions for the global selective search have almost always been territory score based [4,14], i.e. they estimate the territory score or expected territory score. The global search tries to maximize this (expected) territory score. If the program were involved in a dollar a point gambling match, this kind of score evaluation strategy would make sense. But in a tournament play, the reward of a Go match depends on win/loss outcome only, regardless by how many points. Winning by one point is just as good as winning by one hundred points. Thus, programs should maximize the probability of winning [10]. We propose that a position evaluation function should evaluate the chance of winning rather than territory score and that the global search should mini–max the chance of winning. For example, in Fig. 1, taken from a computer Go tournament match, Black should play at point ‘‘b’’ to make sure the Black group at the left hand side of the board will survive, which essentially guarantees a win for Black, since it already has about 20 points lead over White. But the author’s program Go Intellect, based on territory score evaluation in a global selective search, decided to play at point ‘‘a’’, which seemed to be able to generate more territory within the search horizon. Eventually the critical group was killed and the game was lost.

Human players make move decisions based on the chance of winning all the time. For example, when we are ahead, we will reduce the uncertainty by avoiding invasion and fighting moves. But a territory score evaluation based

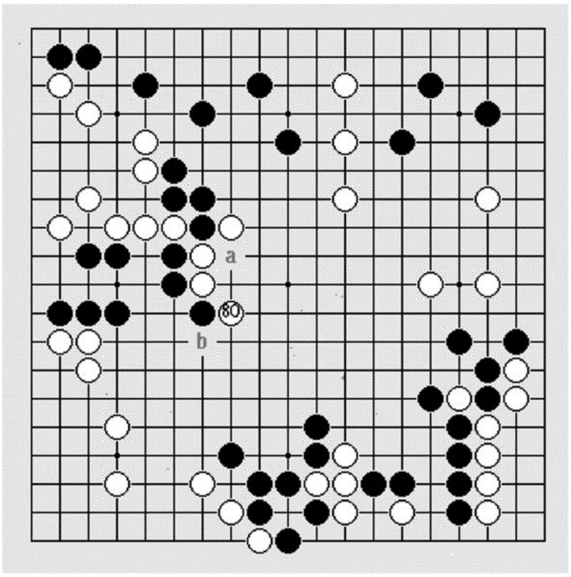

Fig. 1. A tournament game at a FOST Cup Computer Go Championship.

program will easily make a strategic mistake, when it is already comfortably ahead, by invading into the opponent's territory since the territory evaluation after the invasion may appear that the opponent's territory is drastically reduced.

In Sections 2–5, we develop an evaluation function estimating the probability of winning to be used by global selective search based Go programs in finding the move offering the best chance of winning.

Decomposition Search [13] and Soft Decomposition Search [8,9] use territory evaluation scores of the component games to find an optimal move based on Combinatorial Game Theory. In Section 6, we propose a new "Probabilistic Combinatorial Games" model, which can deal with probabilistic outcome distributions of the component games. And we describe an algorithm for playing such probabilistic combinatorial games to maximize the chance of winning. In Section 7, we present the conclusion.

2. Expected territory vs. chance of winning

We shall illustrate the difference between the expected territory and the chance of winning. Let us assume there are k groups on the board. For $i = 1, 2, \ldots, k$, group i probability p_i to live and has a territory effect A_i, which includes the group's area plus half of the neutral surrounding area and A_i is negative if it is of the opponent's color. If there is no other territory to compete and fates of all groups are independent, which are, of course, rather strong assumptions, then the expected territory for us is

$$E_A = \sum_{i=1}^{k}[(p_i * A_i) + (1 - p_i) * (-A_i)] = \sum_{i=1}^{k}(2p_i - 1) * A_i$$

This value is easy to compute. Most evaluation functions for global search in Go more or less return this type of expected territory score. The chance of winning, on the other hand, is calculated as the sum of all probabilities of positive outcomes: $E_W = \sum\{q_1q_2\ldots q_k|\sum_{i=1}^{k}A'_i > 0$, for each $I = 1, 2, \ldots, k$: q_i is either p_i or $1-p_i$, and (A'_i is A_i if $q_i = p_i$; otherwise A'_i is $-A_i$)$\}$.

This is quite different from E_A. Since most groups are safe, i.e. $p_i = 1$ for most $i = 1, 2, \ldots, k$, we do not need to calculate the products $q_1q_2 \ldots q_k$ with $q_i = 1 - p_i = 0$. Instead of 2^k products to compute and add, there are really only 2^{k1} products to add where $k1$ is the number of unsettled groups. So, the computation is not as bad as it first appears.

In addition to groups on the board, there are almost always some no man's lands to compete and the boundaries of the groups can change. A Go program makes territory estimate taking into account, an estimation on no-man's land based on influence values. Fig. 2 is an example of territory estimation by Go

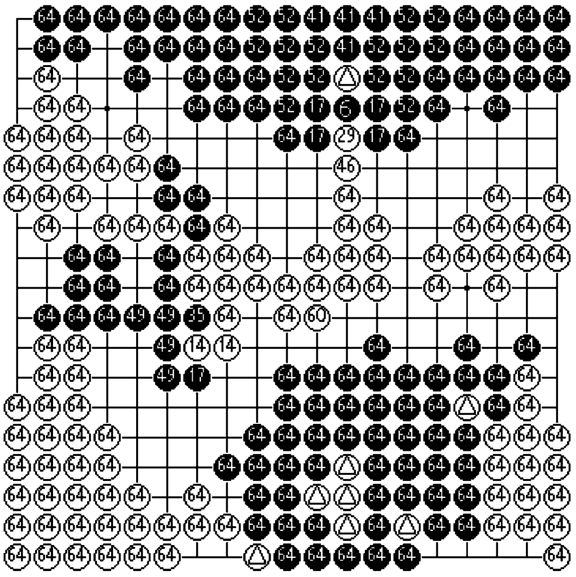

Fig. 2. Territory estimation by Go Intellect. 64 is used as a unit representing full ownership. Dark circles represent Black's territory and light circles represent White's territory, except those circles marked by triangles representing dead stones. Stones marked by triangles are considered dead; they completely belong to the opponent. Numbers less than 64 mean partial ownership of the points. Circles appear at stones as well as some empty points.

Intellect. The number 64 on a grid point means complete control of the point by the color shown. A number less than 64 represents partial ownership. The estimate comes from the sum of all individual point estimates, adjusted by the presence of good and bad features and the komi, where the komi is a predetermined number of points to be deducted from the first player, Black, for balancing the advantage of initiative.

Our heuristic estimate of the chance of winning should also take no-man's land into consideration. In Section 3, we shall discuss probability of winning when there are no unsettled groups on the board. In Section 4, we deal with the case when there are unsettled groups on the board.

3. All groups are safe

In this case, the territory score $S = E_A = \sum_{i=1}^{k} A_i$ is a good prediction of the outcome of the game when it is close to the end of the game, especially when S is large. It is less reliable, when the game is far away from the end—there are too many no man's lands to compete and group boundaries may change. The number of moves played so far is not a good estimator on how close it is to the game end, because the total number of moves in a Go game usually varies from 100+ to 300+. We have found that the total number of frontier space points of all groups is a good indicator on how close it is to the game end. A space point of a group is called a frontier space point if it is adjacent to a point not in the same group [2,11]. Frontier space points indicate that group boundaries are not finalized. Fig. 3 shows the frontier space points, which are marked by X. There are a total of 55 frontier space points on the board, which indicates it is still far away from the game's end. The board configuration is the actual continuation from Fig. 1 at a FOST Cup competition. Black now suffered bad consequence from the earlier strategic mistake, which could easily have been avoided by using the chance of winning evaluations in global search.

The total number of frontier space points is 0 at the beginning, since there are no groups on the empty board. It gradually increases and peaks in mid-game, then gradually decreases to 0 toward the end of the game. When there are only dummy points left on the board, all space points of groups are completely surrounded by their stones and there will be no frontier space points. At that stage, the game has practically ended. Based on the analysis of 36 recent computer Go tournament game records, we have found that it usually takes about 2–4 moves (plies) to reduce a frontier space. After about move 100, the number of moves needed to reach the practical end of the game is roughly three times total number of frontier space points. Let $S = E_A$, $F =$ the total

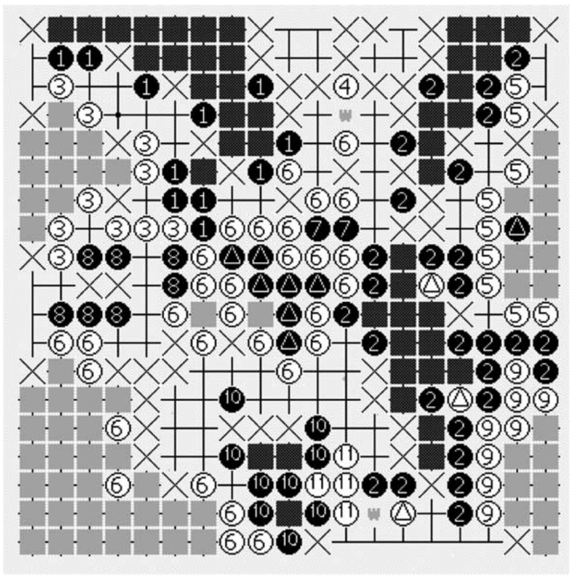

Fig. 3. Frontier space points are marked by X's.

number of frontier space points, and M = number of moves played by both sides so far. Then the chance of winning E_W can roughly be estimated as follows.

```
if ($M < 100$) {
   if ($S > 60 + (100 - M)/4) E_W = 1$;
   else if ($S < -60 - (100 - M)/4) E_W = 0$;
   else $E_W = 0.5 + S/(60 + (100 - M)/4) * 0.5$;
}
else {
   if ($S > F) E_W = 1$;
   else if ($S < -F) E_W = 0$;
   else $E_W = 0.5 * (1 + S/F)$
}
```

The average number of frontier spaces at move 100 is about 60, so we do not expect big shift in the value of E_W from move 99 to move 100. We use $E_W(S, F, M)$, or simply $E_W(S)$, to denote the E_W as determined by the above simple linear approximation. For example, if the game has played 130 moves, and there are 40 frontier space points, and the score estimate is 20 ahead in our favor, then E_W (20, 40, 130) = 0.75. Our chance of winning is estimated to be 75%. This estimate is for computer programs, in particular for the author's program Go Intellect. For human experts, 20-point lead at move 130 would mean almost a sure win.

4. Existence of unsafe groups

Usually there are some unsafe group on the board during opening and midgame. Assume that among all groups on the board, $\{G_i | i = 1, 2, \ldots, k\}$, the first $k1$ groups are safe and the last $k-k1$ groups are unsafe. Then the pessimistic evaluation can be defined as $S_p = \sum_{i=1}^{k1} A_i - \sum_{i=k+1}^{k} |A_i|$, assuming all our unsafe groups will be dead and all opponent's unsafe groups will be alive. Similarly, we define optimistic evaluation to be $S_0 = \sum_{i=1}^{k1} A_i + \sum_{i=k+1}^{k} |A_i|$. The actual outcome should be in between pessimistic evaluation and optimistic evaluation. If we backup S_p in look-ahead search, the program will play very conservatively. On the other hand, If we backup S_0 in look-ahead search, the program will play very aggressively. Michael Reiss' program Go4++ seems to use the strategy of maximizing S_p. Since it uses this strategy from the beginning of the game, it works well with the program's superior territory surrounding capabilities. When S_p is positive or near positive, it is a good strategy to maximize S_p, which guarantees winning. When S_0 is negative or near negative, it is a good strategy to maximize S_0, which represents the only chance of winning. Backing up the average of S_p and S_0 is kind like backing up the expected territory, which is flawed. In the next section, we propose to analyze possible outcomes of battles of unsafe groups as the base of chance of winning evaluation.

5. Battles

We shall ignore unsafe groups with values less than what a passive territory-surrounding move can gain; we treat them as abandoned stones. If there is a valuable group that is unsafe, then the transitive closure of adjacent unsafe groups of both colors form a battle [8,9]. Fig. 4 shows a battle.

The fates of the groups in a battle are obviously mutually dependent. The outcome of a battle frequently decides the win or loss of the whole game. So we opt to evaluate the whole board instead of individual groups. The best outcome for us is that all our unsafe groups become safe and all the opponent's unsafe groups become dead. The worst outcome is that all unsafe groups in the battle become opponent's territory. Usually there are many in-between outcomes. Assume there is a probability p_i for outcome score S_i for $i = 1, 2, \ldots, n$ ($\sum_{i=1}^{n} p_i = 1$). Then the chance of winning is heuristically estimated to be $\sum_{i=1}^{n} p_i * E_\mathrm{W}(S_i)$. On a rare occasion, there are multiple, simultaneous battles.

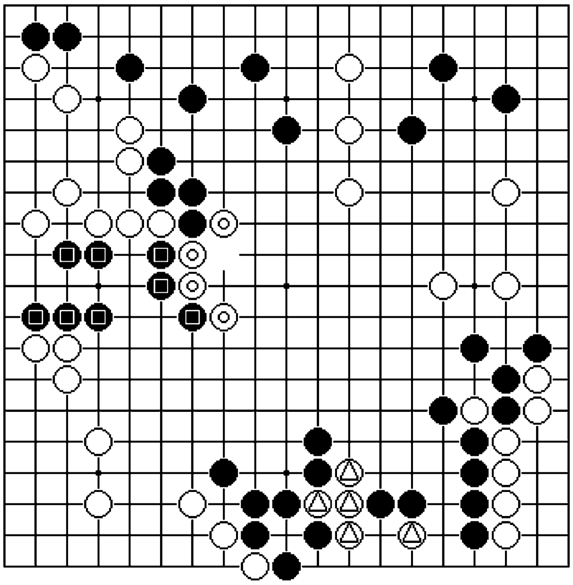
Fig. 4. The Black group marked by squares and the White group marked by small circles form the main battle of the game configuration. The White stones marked by triangles are considered abandon stones.

Then we have to take into account what will happen if the opponent gets initiative in a battle as well. The global selective tree search model is no longer adequate in this case [6]. To handle that type situations adequately, we need a probabilistic combinatorial game model and meta-search for selecting the move offering the best chance of winning, which we shall present in the next section.

6. Probabilistic combinatorial game (PCG) model and the meta-search technique

Let us define an outcome distribution as a finite set of ordered pairs of the form (S,p), where S is an integer score and p is the probability that score S will be the result, $0 < p <= 1$, such that the sum of all the p's equals 1, i.e. an outcome distribution $D = \{(S_i,p_i) | S_i \in I,\ 0 < p_i <= 1, \text{ and } \sum_{i=1}^{n} p_i = 1\}$. Next we define probabilistic combinatorial games recursively as follows:

1. An outcome distribution is a PCG.
2. A sum of PCGs is a PCG.
3. $\{A_1, A_2, \ldots, A_n | B_1, B_2, \ldots, B_m\}$ is a PCG if $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m$ are PCGs. $A_1, A_2, \ldots, A_n$ are called the Left options and $B_1, B_2, \ldots B_m$ are called the Right options, just like in the ordinary combinatorial games.

Intuitively, we can think of a PCG in the following way: the playing rules are the same as in ordinary combinatorial games [1] except when a sub-game reaches an outcome distribution, dice will be thrown to select an outcome of the sub-game according to the probability distribution. And the game continues with the outcome of this sub-game known to both sides. If the final total outcome scores of all component sub-games is positive, it is a win for Left, otherwise it is a win for Right.

In Go, after decomposition [12,13] or soft-decomposition [8,9], each battle and other local component game can be represented by a component PCG, G_i, and the whole game is represented by $G = \sum_{i=1}^{n} G_i$ assuming that there are n sub-games. If, for each outcome distribution $D = \{(S_i,p_i) | S_i \in I,\ 0 < p_i <= 1, \text{ and } \sum_{i=1}^{n} p_i = 1\}$, we use $e = \sum_{i=1}^{n} p_i * S_i$ to replace D in G, then we get an ordinary combinatorial game model. We can use the following complete Meta-search to find the move that will produce the highest expected territory score, with k representing the score of the settled portion of the game.

```
int ExpectedScore ($\sum_{i=1}^{n} G_i$, $k$, toPlay, bestMove) {
   bestMove = Empty;
   if ($n$ = 0) return $k$;
   else if (none of $G_i$ is a number) {
      if (toPlay = L) {
```

```
      bestMove = the move s.t. ExpectedScore ($\sum_{j=i} G_j + G_i^L$, $k$, $R$, bM) is
      max;
      return max (ExpectedScore ($\sum_{j!=i} Gz_j + G_i^L$, $k$, $R$, bM));
    }
    else {
      bestMove = the move s.t. ExpectedScore ($\sum_{j!=i} G_j + G_i^R$,$k$, $L$, bM) is min;
      return min (ExpectedScore ($\sum_{j!=i} G_j + G_i^R$, $k$, $L$, bM));'
    }
  }
  else

  return ExpectedScore ($\sum_{j!=i} G_j$, $k + k_i$, toPlay, bestMove); // $G_i$ is the first
  number in G.
}
```

The above function procedure will maximize the expected score. Hence, if we play for a dollar per point, this is the procedure to use. As discussed earlier, in a normal tournament setting, we want to maximize the chance of winning. We can use the following recursive procedure to achieve the optimal chance of winning. We assume that there are no outcome distributions at the top level. We will use k to represent the score of the settled portion of the game. The following meta-search procedure finds the optimal move that has the best chance of winning and returns its winning chance.

```
float WinningProbability ($\sum_{i=1}^{n} G_i$, $k$, toPlay, bestMove) {
  bestMove = Empty;
  if ($\sum_{i=1}^{n} G_i = \phi$) {
    if ($k > 0$) return 1;
    else return 0;
  }
  else if (none of $G_i$ is an outcome distribution) {
    if (toPlay = L) {
      bestMove = the move s.t. WinningProbability ($\sum_{j!=i} G_j + G_i^L$, $k$, $R$, bM)
      is max;
      return max (WinningProbability ($\sum_{j!=i} G_j + G_i^L$, $k$, $R$, bM));
    }
    else {
      bestMove = the move s.t. WinningProbability ($\sum_{j!=i} G_j + G_i^R$, $k$, $L$, bM)
      is min;
      return min (WinningProbability ($\sum_{j!=i} G_j + G_i^R$, $k$, $L$, bM));
    }
  }
  else
```

```
    return $\sum_{l=1}^{m} p_l$ *WinningProbability ($\sum_{j!=i} G_j$, $k + S_l$, toPlay, bestMove); // $G_i = \{(S_l, p_l) | l = 1, 2, \ldots, m\}$ is the first outcome distribution in $G$.
}
```

7. Lessons learned from implementation and testing

The above strategy of evaluating the chance of winning is valid for games that count scores in deciding the winner, and have uncertainty involved in the score counting for non-terminal positions, such as Go, Amazon, Domineering, … etc. The global evaluation function for the probability of winning has been implemented in an experimental version of Go Intellect. The result is mixed. It works fine when there are no battles on the board. But when there are one or more battles on the board, the probabilities p_i and the corresponding outcomes scores S_i are difficult to be estimated with good accuracies by the machine. In this case, the experimental version of Go Intellect had a slightly inferior performance than the regular version Go Intellect, which estimates the territory scores. The problem was that the evaluation function for the probability of winning was not good enough when there are unsettled groups on the board. More thorough knowledge engineering and implementation will be needed to take the advantage of this approach of maximizing the chance of winning. In Go, features that are easy to compute by machine, such as number of liberties of a block, or Manhattan distance of a stone to a safe friendly group, generally have less impact to the final outcome of a game; features more closely associated to the final outcome of a game, such as probability of winning, or result of a battle, tend to be elusive and hard for the machine to grasp. Significant research effort will be needed to find a good handle on estimating the chance of winning and the probability distribution of outcomes.

Instead of using an evaluation function returning the probability of winning directly, we have found the following two techniques using the probability of winning indirectly to be very beneficial to a Go program which is based on global search for maximizing the (expected) territory-score.

Technique I. Dynamic modification of weights on some move generators—when the probability of the winning estimate is high, reduce the weights of attack and invade routines, and increase the weights on protect group and protect territory routines; when the probability of winning estimate is low, do the reverse.

Technique II. Adjust territory evaluations by the probability of winning estimate. For example, if the chance of winning is >99%, add 10 points to the territory score in the evaluation function, and if the chance of winning is <1%, subtract 10 points from the territory score. This can avoid some of the pitfalls of territory based global search. A smooth way to adjust territory evaluation may be by adding (probability of winning—0.5) $* k$ for some constant k depending on how confident the program is on its probability of winning estimate.

Implementation of the above two heuristic techniques has incrementally improved the performance of the program.

Since the evaluation function value for Go will have significant inaccuracy, whether we evaluate territory, or chance of winning, or something else linked to the outcome of the game, we need to find an alternative strategy to mini-max for searching Go game tree effectively before a breakthrough in the playing strength of Go programs can happen.

References

[1] E. Berlekamp, J.H. Conway, R.K. Guy, Winning Ways, Academic Press, New York, 1982.

[2] K. Chen, Group identification in Computer Go, Go chapter in the book, in: D. Levy, D. Beal (Eds.), Heuristic Programming in Artificial Intelligence, Ellis Horwood, 1989, pp. 195–210.

[3] K. Chen, Move decision process of go intellect, Comput. Go 14 (1990) 9–17.

[4] K. Chen, Heuristic search in Go Game, in: Proceedings of Joint Conference on Information Sciences '98, vol. II, 1998, pp. 274–278.

[5] K. Chen, Some practical techniques for global search in Go, ICGA J. 23 (2) (2000) 67–74.

[6] K. Chen, A study of decision error in selective game tree search, Inf. Sci. 135 (3–4) (2001) 177–186.

[7] K. Chen, Computer go: knowledge, search, and move decision, ICGA J. 24 (4) (2001) 203–215.

[8] K. Chen, Soft decomposition search in the game of Go, in: Proceedings of the 6th International Conference on Computer Science and Informatics, 2002, pp. 461–464.

[9] K. Chen, Soft decomposition search and binary game forest model for move decision in Go, Inf. Sci. 154 (3–4) (2003) 157–172.

[10] K. Chen, A new positional evaluation strategy for global search in Go, in: Proceedings of 7th Joint Conference on Information Sciences, 2003, pp. 493–496.

[11] A. Kierulf, K. Chen, J. Nievergelt, Smart game board and go explorer: a case study in software and knowledge engineering, Comm. ACM 33 (2) (1990) 152–166.

[12] K. Kao, Mean and temperature search for Go endgames, Inf. Sci. 122 (2000) 77–90.

[13] M. Müller, Decomposition search: a combinatorial games approach to Game Tree Search, with applications to solving Go endgames, IJCAI-99 (1999) 578–583.

[14] M. Müller, Counting the score: Position evaluation in computer Go, ICGA J. 25 (4) (2002) 219–228.