

Soft Decomposition Search in Computer Go

Keh-Hsun Chen

Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA
e-mail: chen@uncc.edu

Abstract

In this paper, a new approach to game tree search in Go, called *soft decomposition search*, is proposed. A simplified model, *binary game forest*, for move selection in Go is presented. A new paradigm for move decision in computer Go based on soft decomposition search and binary game forest is introduced.

Keywords: computer Go, combinatorial game theory, soft decomposition search, binary game forest, move-decision strategy

1. Introduction

Decomposition search [Mueller 1999 & 2001] involves many local searches, but the entire process is for global move selection. It has sound theoretical foundation in combinatorial game theory [Berlekamp & al. 1982]. It consists roughly the following four steps: 1. game decomposition & sub-game identification, 2. local combinatorial game searches, 3. simplification of combinatorial game expressions, and 4. sum game play. Decomposition search works well for late stage end-game of Go but is not suitable for earlier stage of the Go game. In 1998, the author independently implemented the idea of combinatorial game theory in his preliminary version of Go Intellect II. It performed the steps 1 & 2 of the decomposition search and similar game selection strategy to the steps 3 and 4. The result was a disaster – it dropped from the second place finish, of Go Intellect I, in the FOST Cup in 1997 to the sixth place finish in the same event in 1998. The program was weaker than its global selective search based predecessor. We have found two major reasons for the failure of the decomposition search:

Candidate moves sometimes need to be outside the pre-decomposed region after a few moves of look-ahead and was artificially suppressed because they were not in the marked area.

The decomposition of the board into sub-regions do not really produce “independent “ sub-games. The development of a region may have significant impact to some other regions, which is not measured in the local sub-game score.

The *soft decomposition search* proposed in this paper will fix these two problems. It uses the concept of core groups to identify sub-game instead a pre-specified region, which allows key candidate moves to be generated as needed without the restriction of pre-set region bound, just like a human Go player would do. It uses global evaluation for each terminal node of a local search, so the impact to the whole board can be measured.

In the next section, we shall introduce a simplified combinatorial game theory model, binary game forest, for the soft decomposition search in Go. Soft decomposition and associated local searches will be described in section 3. Section 4 discusses basic

properties of binary game forests. Section 5 develops a dominance relation on binary game trees for simplifying move selection processes. Section 6 introduces three heuristic move selection algorithms based on binary game forest model. In section 7, we propose a meta-search approach on the binary game forest model as a new paradigm for move decision in Go.

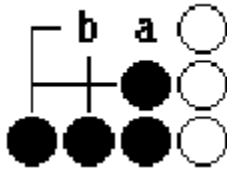
2. Binary game forest model

In this section, we shall introduce the binary game forest (BGF) model, which consists of a set of binary game trees (BGTs). This model can be used for any game with the property that the best move of a component game can be determined locally within that component without referring to other components of the game. This basic assumption generally holds for Go game situations although not always so. It constitutes a simplified approximation model for Go. The greatest gain in using this model is the reduction of the computational complexity in move decision.

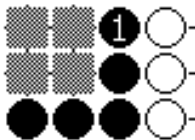
2.1. Binary game trees and components of a Go board configuration

Following the notation of [1], we shall call Black the Left player L and White the Right player R. L favors plus scores and R favors minus scores. Before we formally define BGT and BGF, let's examine some local game situations. Japanese scoring method will be used throughout the discussion. We assume this component does not involve any life/death problems.

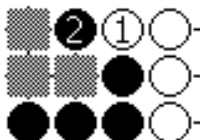
Example 2.1.1.



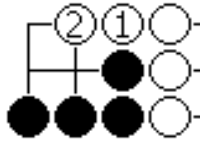
If L(Black) is to play in this game component, he will play at a to surround 4 points:



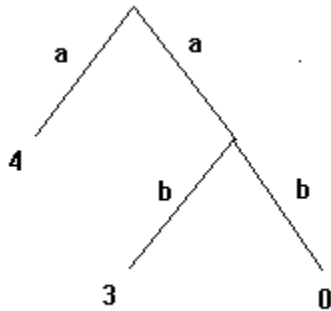
If R(White) is to play, he will play at a also then L will block at b. The result is 3 points for L.



If L ignores R's initial move at a, then R can continue play at b and the point count will become 0.

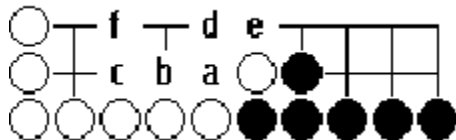


This situation can be represented by the following binary game tree T1:

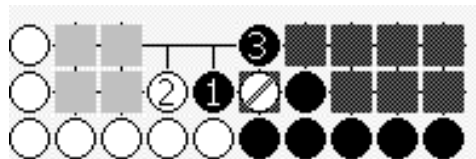


In combinatorial game theory notation, $T1 = \{ 4 \parallel \{ 3 \mid 0 \} \}$. Left branches represent L (Black)'s moves and right branches represent R(White)'s moves. Terminal nodes are marked with resulting scores. (Edges are marked with corresponding move locations). If we use Chinese scoring method instead, the only difference will be a shift on the score marks on the terminal nodes.

Example 2.1.2.

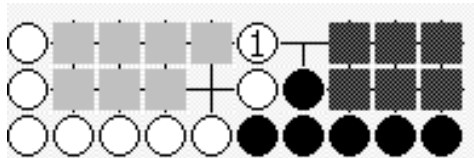


Again we assume there are no Life/Death problems here. If Black is to play, a will be its best move, after Ba, Wb, Be, Black has 9 points and White has 4 points as marked in the next diagram (Note that a prisoner is worth one extra point).

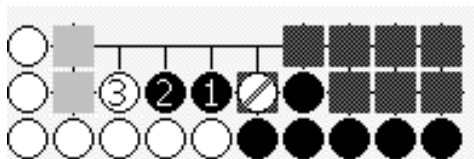


A net of 5 points is to Black (L). Hence the score is +5. If after Ba, Wb, Black skips a move (plays at some other component), then White can play at d. The result will net to one point to White's favor (hence the score -1).

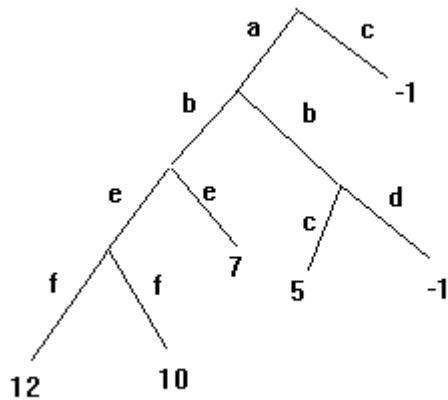
If White is to play first, the score will be $6 - 7 = -1$ as shown in the following diagram:



When Black gets to play first 2 moves consecutively, we have



This component game can be represented by BGT T2:



$$T2 = \{ \{ \{ \{ 12 \mid 10 \} \parallel 7 \} \parallel \{ 5 \mid -1 \} \} \parallel -1 \}$$

In early stage of a Go game, the search won't reach true terminal nodes. We will use static evaluation function values to mark temporary terminal nodes.

Now we are ready to give a formal definition of BGT.

Definition 2.1.1. A binary game tree is a binary tree such that.

- 1) Every non-terminal node has 2 successors - a left successor and a right successor.
- 2) Every terminal node is associated with a rational number (usually an integer).

If T is a single node BGT, we denote it by the number n , which marks the only terminal node. If T is not a single node BGT, we shall use the notation $T = [T^L|T^R]$, where T^L is the left subtree of T & T^R the right subtree of T .

Convention. In case T is a single node game tree both T^L & T^R denote T itself.

We shall define the left value and the right value of a BGT T recursively as follows:

Definition 2.1.2.

$$\begin{aligned}
 L_v(T) &= v && \text{if } T \text{ is a single node BGT with node value } v. \\
 &R_v(T^L) && \text{otherwise.} \\
 R_v(T) &= v && \text{if } T \text{ is a single node BGT with node value } v. \\
 &L_v(T^R) && \text{otherwise.}
 \end{aligned}$$

So $L_v(T)$ (resp. $R_v(T)$) represents the outcome of the component game represented by T assuming that L (resp. R) plays first & each player alternately plays on the component without skipping a move until a terminal value is reached.

We shall assume, in our model, each non-singleton binary game tree/subtree T has a $L_v(T) > R_v(T)$; otherwise no player will want to play on that component first. We can replace a T (could be a subtree) having $L_v(T) \leq R_v(T)$ by a single node with the marked value determined by the rules of the game represented.

2.2. Binary game forest model.

A full board Go game can be partitioned into a set of components. Each of these components can be represented as a binary game tree. Hence a whole Go game can be represented as a collection of BGTs.

Definition 2.2.1. A binary game forest (BGF) F is a set of binary game trees. In notation,

$$F = \{ T_1, T_2, T_3, \dots, T_n \} = T_1 + T_2 + T_3 + \dots + T_n = \sum_{i=1}^n T_i$$

where each T_i is a BGT.

Typically a Go game configuration can be represented by a BGF consisting of a dozen or so BGTs. Many of these BGTs will be obviously “too small” relatively, and can be omitted for current move decision purpose. We shall discuss the model set-up process in the next section.

3. Soft decomposition

There are two types of sub-games in a Go Board configuration: urgent sub-games and calm sub-games. An urgent sub-game always involves one or more unsafe groups. A calm sub-game does not involve the safety issue but sometimes may be bigger than an urgent sub-game.

For an urgent sub-game, we start with an unsafe group of either color then find the transitive closure of adjacent unsafe groups. All these related unsafe groups form the core of the sub-problem. Attack/protect moves can be generated for each unsafe group in

the core via pattern libraries or heuristics. Various local selective searches are performed for each urgent sub-game, which normally include at least what happens if we play first, what happens if the opponent plays first, what happens if we play first and the opponent ignores, and what happens if the opponent plays first and we ignore. The alpha-beta back-ups and the principal variations can be used to construct a BGT such as:

$$T = \{ \{ N_{LL} | N_{LR} \} || \{ N_{RL} | N_{RR} \} \}$$

We note that the local searches may go pretty deep but the representing BGT for the sub-game can usually stop at height 2. For calm sub-games, we can just consider moves that may surround more territory of ours or reduce more territory of the opponent's. Shallow look-ahead searches usually suffice for a calm sub-game. Some sub-games are obviously insignificant comparing to some others in the current board configuration through heuristic estimates and we can omit the construction of the corresponding BGTs. Collect those more significant BGTs to form the BGF for the current board configuration. We shall discuss how to make the move decision based on the BGF in sections 6 and 7 after we establish some basic properties of BGFs in section 4 and a dominance relation on BGTs in section 5.

4. Basic properties of binary game forests

Definition 4.1 We define

$$L_{\text{choice}}(k, \Sigma_{i=1}^n T_i) = \begin{cases} \Sigma_{i=1}^{k-1} T_i + T_k^L + \Sigma_{i=k+1}^n T_i & \text{if } 1 \leq k \leq n \\ \Sigma_{i=1}^n T_i & \text{otherwise} \end{cases}$$

$$R_{\text{choice}}(k, \Sigma_{i=1}^n T_i) = \begin{cases} \Sigma_{i=1}^{k-1} T_i + T_k^R + \Sigma_{i=k+1}^n T_i & \text{if } 1 \leq k \leq n \\ \Sigma_{i=1}^n T_i & \text{otherwise} \end{cases}$$

$L_{\text{choice}}(k, \Sigma_{i=1}^n T_i)$ is the status of $\Sigma_{i=1}^n T_i$ after L plays at T_k . $R_{\text{choice}}(k, \Sigma_{i=1}^n T_i)$ is the status of $\Sigma_{i=1}^n T_i$ after R plays at T_k . If k is not in the range of 1 to n , the BGF does not change i.e. the player passes.

When each BGT in a BGF F is a single node BGT, we say that F is game-over. The left (resp. right) outcome L_o (resp. R_o) of a BGF is defined as the scoring result after optimal sequence of play starting with L (resp. R).

Definition 4.2

$$L_o(\Sigma_{i=1}^n T_i) = \begin{cases} \Sigma_{i=1}^n L_v(T_i) & \text{if } \Sigma_{i=1}^n T_i \text{ is game-over.} \\ \text{Max}_{0 \leq k \leq n} R_o(L_{\text{choice}}(k, \Sigma_{i=1}^n T_i)) & \text{otherwise} \end{cases}$$

$$R_o(\Sigma_{i=1}^n T_i) = \begin{cases} \Sigma_{i=1}^n R_v(T_i) & \text{if } \Sigma_{i=1}^n T_i \text{ is game-over.} \\ \text{Min}_{0 \leq k \leq n} L_o(R_{\text{choice}}(k, \Sigma_{i=1}^n T_i)) & \text{otherwise} \end{cases}$$

Definition 4.3 Let $F = \Sigma_{i=1}^n T_i$ be a BGF. We define $F^L = L_{\text{choice}}(k, \Sigma_{i=1}^n T_i)$ where k is the smallest integer with the property that $L_o(\Sigma_{i=1}^n T_i) = R_o(L_{\text{choice}}(k, \Sigma_{i=1}^n T_i))$. We define F^R in a similar way with R & L swapped.

We shall define the negative of a BGT and the negative of a BGF in the following two definitions.

Definition 4.4 Let T be a BGT. If T is a single node BGT with node value v ; define $-T =$ a single node BGT with node value $-v$, otherwise define $-T = [-T^R|-T^L]$.

Theorem 4.1 For any BGT T , we have $L_o(T + (-T)) = R_o(T + (-T)) = 0$.

proof. By induction on $h =$ the height of BGT T .

We shall call $-T$ the negative of T . $T - T'$ shall mean $T + (-T')$.

Definition 4.5 Let $F = \sum_{i=1}^n T_i$ be a BGF. We shall define the negative of F to be $-F = \sum_{i=1}^n -T_i$. And $F - G$ shall mean $F + (-G)$.

The following five basic theorems about BGFs from [Kao & Chen 1992] are needed for the development of the dominance theory in section 5.

Theorem 4.2 For every BGF F , $L_o(F) \geq R_o(F)$.

Theorem 4.3 For every BGF F_1, F_2 ,

$$L_o(F_1) + L_o(F_2) \geq L_o(F_1 + F_2) \geq L_o(F_1) + R_o(F_2)$$

$$R_o(F_1) + R_o(F_2) \leq R_o(F_1 + F_2) \leq R_o(F_1) + L_o(F_2)$$

Theorem 4.4 For every BGF F , $L_o(F + (-F)) = R_o(F + (-F)) = 0$

Theorem 4.5 For every BGF F , $L_o(F) = -R_o(-F)$.

Theorem 4.6 For every BGF F, G ,

$$L_o(F + G - G) = L_o(F)$$

$$R_o(F + G - G) = R_o(F)$$

Now we shall define a partial ordering \geq on BGFs based on outcome values. Let F, G denote any two BGFs.

Definition 4.6 $F \geq G$ if and only if for every BGF X , $L_o(F+X) \geq L_o(G+X)$ and $R_o(F+X) \geq R_o(G+X)$.

Obviously, \geq is reflexive and transitive. i.e. $F \geq F$ and $F_1 \geq F_2$ & $F_2 \geq F_3$ implies $F_1 \geq F_3$.

Definition 4.7 $F \leq G$ if and only if $G \geq F$.

Definition 4.8 $F = G$ if and only if $F \geq G$ & $F \leq G$.

This makes \geq anti-symmetric. Definitions 4.6, 4.7 & 4.8 can be applied to BGTs as well, viewing a BGT T as a BGF with only one component BGT. We shall use a number k to denote a BGT of a single node with the associate value k , or even a BGF with such a BGT alone. We shall omit the proofs of the following theorems:

Theorem 4.7 $F \geq k$ if and only if $R_o(F) \geq k$.

Theorem 4.8 $F \leq k$ if and only if $L_o(F) \leq k$.

Corollary 4.8.1 For any BGT T , $R_v(T) \leq T \leq L_v(T)$ and for any BGF F , $R_o(F) \leq F \leq L_o(F)$.

Corollary 4.8.2 $G \leq 0$ if and only if $L_o(G) \leq 0$

Corollary 4.8.3 $G = 0$ if and only if $G \geq 0$ & $G \leq 0$

Corollary 4.8.4 $F - F = 0$.

Theorem 4.9 $F_1 \geq G_1$ & $F_2 \geq G_2$ implies $F_1 + F_2 \geq G_1 + G_2$

Corollary 4.9.1 $F \geq G$ implies $F + X \geq G + X$.

Corollary 4.9.2 $F - G \geq 0$ if and only if $F \geq G$

Theorem 4.10 $T^L \geq T \geq T^R$

Theorem 4.11 $T_1 \geq T_1'$ & $T_2 \geq T_2'$ implies $[T_1 | T_2] \geq [T_1' | T_2']$

5. Dominance Relation

In this section, we shall develop a dominance theory for BGTs in BGFs. This will be the foundation of a computationally efficient & near optimal solution method for BGFs proposed in Section 6.

Definition 5.1. Let T_1 and T_2 be BGTs. We say T_1 dominates T_2 , in notation $T_1 \gg T_2$, if and only if $T_1^L + T_2 \geq T_1 + T_2^L$ and $T_1^R + T_2 \leq T_1 + T_2^R$.

This means playing at T_1 is at least as advantageous as playing at T_2 for both L and R in any BGF including T_1 & T_2 . The dominance relation \gg is reflexive, transitive but not anti-symmetric. We note that $T_1 \gg T_2$ implies $T_1^L - T_1^R \geq T_2^L - T_2^R$ but the converse is not true.

Definition 5.2. $T_1 = T_2$ if and only if $T_1 \gg T_2$ & $T_2 \gg T_1$

$T_1 = T_2$ means that it makes no difference in a BGF including T_1, T_2 whether to play at T_1 first or at T_2 first. The following example shows \gg is not anti-symmetric w.r.t. =.

Example 5.1. $[5 | -5] = [10 | 0]$ but $[5 | -5] \neq [10 | 0]$.

A number (i.e. single node BGT) is always dominated by any BGT. Since the outcome in that component has already been decided, a player is better off to play at somewhere else. The following theorems can be established.

Theorem 5.3 $T \gg k$ where T is a BGT and k is a number (a singleton BGT).

Theorem 5.4. $[a | b] \gg [c | d]$ if and only if $a-b \geq c-d$.

We shall use $L_{\sim v}(T)$ to denote the node value of $T^{LLL\dots}$ and $R_{\sim v}(T)$ to denote the node value of $T^{RRR\dots}$.

Definition 5.3. Let T be a BGT. The inner swing of T is defined to be $[L_v(T) | R_v(T)]$ and the outer swing of T is defined to be $[L_{\sim v}(T) | R_{\sim v}(T)]$.

Note that the outer swing of T always \gg the inner swing of T .

Theorem 5.5. Let $[c|d]$ be the outer swing of T . Then $[c|d] \gg T$.

Theorem 5.6. Let T be a BGT of height ≤ 3 and $[a|b]$ be the inner swing of T . Then $T \gg [a|b]$.

If the height of a BGT T is greater than 3, Theorem 5.6 does not always hold. For example,

Example 5.2. Let T be the following BGT of height 4:

$$[[[7|6] | [5 | [3|-100]]] | [[2|1] | 0]].$$

Then C is $[5 | [3|-100]]$ and the inner swing of T is $[5|1]$ and $-T$ is $[[0 | [-1|-2]] | [[[100|-3] | -5 | [-6|-7]]]$. Readers can draw

BGTs to verify $L_o(R_{\sim v}(T) - T + [a|b] - a) = -1$. Hence

NOT $(T^L - T + [a|b] - a \geq 0)$ i.e. NOT $(T^L + [a|b] \geq T + a)$ So, NOT $(T \gg [a|b])$.

Nevertheless, $T \gg$ inner swing of T is generally true. Even when it is not true, it is not far off.

6. Some fast approximate solutions to binary game forests

In the following discussion, we shall assume we are the left player L. Given a BGF $F = \sum_{i=1}^n T_i$, we would like to find a way to decide which component to play next to get the optimal result, i.e. determine a k s.t. $L_o(\sum_{i=1}^n T_i) = R_o(L_{\text{choice}}(k, \sum_{i=1}^n T_i))$

If we try an exhaustive meta-search on BGF, since the game will last $h = \sum_{i=1}^n \text{height}(T_i)$ moves and each move will involve usually n choice, it will take $O(n^h)$ computing time. So it is still an intractable problem, it is feasible only when n and h are small. In this section, we shall propose some linear time heuristic algorithms for approximate solutions.

Definition 6.1. Let $F = \sum_{i=1}^n T_i$ be a BGF. The basic value of the move k (i.e. the left move on T_k) is define to be the size of the inner swing of T_k . i.e.

$$\text{Basic value of move } k = L_v(T_k) - R_v(T_k).$$

The notion of sente is very important in Go. Intuitively, sente move means that opponent needs to respond to the move in the same component game, all moves in other components will result in an inferior, or at best equal, result for the opponent. So rigorously, the move at component k by L is sente if and only if

$$R_o(\sum_{i=1}^{k-1} T_i + T_k^L + \sum_{i=k+1}^n T_i) = L_o(\sum_{i=1}^{k-1} T_i + T_k^{LR} + \sum_{i=k+1}^n T_i)$$

It is essentially as hard to figure out if a move is sente from this formula as to figure out the best move from the mini-max equations of L_o and R_o . More practically, we shall use the following definition of sente:

Definition 6.2. Let $F = \sum_{i=1}^n T_i$ be a BGF. The L-sente value of the move k is defined to be the distance of the inner swing of T_k^L . i.e. L-sente value of move $k = LL_v(T_k) - L_v(T_k)$, where $LL_v(T)$ denote the node value of $T^{LLRLR\dots}$. A move k is an L-sente if and only if its L-sente value \geq the basic value of move i for all $i \neq k$. We can define R-sente value and R-sente in a similar way.

Note that Given a BGF, the inner and outer swings and the set of all L-sente/R-sente moves and their L-sente/R-sente values can be computed in linear time to $h = \sum_{i=1}^n \text{height}(T_i)$, hence linear time to n if the heights of BGTs are given a constant bound.

Next, we shall propose some linear time heuristic algorithms for solving BGF $F = \sum_{i=1}^n T_i$. Again, assume L is to play.

Heuristic Algorithm 1:

If there is a move which is both L-sente and R-sente then
 play such a move with highest basic move value
elsif there is an L-sente move then
 play such a move with highest basic move value
else
 play the move with highest basic move value

Heuristic Algorithm 2:

Play the move with highest combined move value = basic value + 1/2 L-sente value

Heuristic Algorithm 3:

Play the move with highest combined move value = basic value + 1/2 L-sente value + 1/4 R-sente value

Lacking theoretical analysis, the effectiveness of these algorithms will be reported after experimenting in a Go playing program.

7. Meta-search

In this section, we shall discuss how to simplify a BGF and how to evaluate a BGF. Using them along with the dominance theory developed in Section 5, we shall

propose a polynomial time heuristic meta-search algorithm that can find an optimal/near optimal move to a BGF in real time, which in turn will provide a sound move decision for the original Go board configuration.

A BGT can be trimmed without affecting its basic move value: when we prune a subtree at an interior node n , if n is a left (resp. right) child, attach to this newly created terminal node with $R_v(S)$ (resp. $L_v(S)$) where S is the pruned subtree.

Let $F = \sum_{i=1}^n T_i$ be a BGF. $L_o(F)$ can be estimated in the following way:

Step 1. Classify all BGTs in F into 5 subBGFs $F_A, F_B, F_S, F_O,$ and $F_N,$

Where F_A = the set of all BGTs in F that are single node BGTs, F_B = the set of all BGTs in F that are both L-sente and R-sente, F_S = the set of all BGTs in F that are L-sente but not R-sente, F_O = the set of all BGTs in F that are R-sente but not L-sente, F_N = the set of all BGTs in F that are neither L-sente nor R-sente.

Step 2. Let $n_1, n_2, n_3 \dots$ be the node values of BGTs in F_A . Compute $V_A = n_1 + n_2 + n_3 + \dots$

Step 3. Order all inner swings for BGTs in F_B according to \gg :

$$[a_1|b_1] \gg [a_2|b_2] \gg [a_3|b_3] \gg \dots \gg [a_j|b_j]$$

Compute $V_B = a_1 + b_2 + a_3 + b_4 + \dots$

Step 4. Let $[a_1|b_1], [a_2|b_2], [a_3|b_3], \dots, [a_k|b_k]$ be the inner swings of BGTs in F_S . Compute $V_S = a_1 + a_2 + a_3 + \dots + a_k$.

Step 5. Let $[a_1|b_1], [a_2|b_2], [a_3|b_3], \dots, [a_l|b_l]$ be the inner swings of BGTs in F_O . Compute $V_O = b_1 + b_2 + b_3 + \dots + b_l$.

Step 6. Order all inner swings for BGTs in F_N according to \gg :

$$[a_1|b_1] \gg [a_2|b_2] \gg [a_3|b_3] \gg \dots \gg [a_m|b_m]$$

Compute $V_N = a_1 + b_2 + a_3 + b_4 + \dots$

Step 7. Return $V = F_A + V_B + V_S + V_O + V_N$ as the estimate of $L_o(F)$.

$R_o(F)$ can be estimated in a similar way with the modification of using

$V_B' = b_1 + a_2 + b_3 + a_4 + \dots$ and $V_N' = b_1 + a_2 + b_3 + a_4 + \dots$ in places of V_B and V_N .

We are ready to introduce the heuristic meta-search algorithm for solving any BGF F . Depending on the resource, we should trim down the BGTs in F to "reasonable size" using the pruning method mentioned in the second paragraph of this section. Also, we can identify and delete dominated BGTs in F . We can order all meta-moves in the simplified BGF according to the heuristics in parallel to those in the heuristic algorithms 1, 2, or 3 in section 6.

During the alpha-beta type meta-search on BGF, if the total heights of the BGTs is too large for the meta-search to reach a terminal position, it can use the heuristic estimate method mentioned above to provide a very good static evaluation. In this case, if L (resp. R) is to move next at a temporary terminal node, the evaluation uses the estimate for $L_o(F')$ (resp. $R_o(F')$), where F' is the part of F still to be explored in the search at the temporary terminal node.

If the number of BGTs in the BGF is not more than a handful, and total heights of these trees is not larger than a dozen or so, then we can use the following alpha-beta based meta-search procedure, written in a Pascal-like pseudo code, to solve the BGF.

Metasearch(F: BGF; WeToPlay: boolean; var bestTree: integer; α , β : integer): integer;

```
var    i, bestV, bt, v, sum: integer;
      F', F0, F1: BGF;
begin
  bestTree := 0;
  Let F = F0 U F1 where F0 is the set of numbers in F and F1 is the set of
  non-number BGTs in F.
  if (F0 is empty) sum := 0 else sum :=  $\Sigma$  F0;
  if (F1 is empty) return sum;
  if (WeToPlay)
  begin
    bestV :=  $\alpha$  - sum;
    for i := 1 to |F1| do
    begin
      F' := replace Ti by  $Ti^L$  in F1;
      v := Metasearch(F', false, bt, sum+bestV,  $\beta$ );
      if (v > bestV) begin bestV := v; bestTree := i end;
    end
  end
  else
  begin
    bestV :=  $\beta$  - sum;
    for i := 1 to |F1| do
    begin
      F' := replace Ti by  $Ti^R$  in F1;
      v := Metasearch(F', true, bt,  $\alpha$ , sum+bestV);
      if (v < bestV) begin bestV := v; bestTree := i end;
    end
  end
  return sum+bestV;
end
```

This procedure is initially called by

Metasearch(BGF, true, best_choice, -361, 361);

If the total heights of the BGTs in the BGF is too high, for a complete meta-search, we can use heuristic estimate at depth_limit:

Metasearch(F: BGF; WeToPlay: boolean; var bestTree: integer; α , β , d: integer): integer;

```

var    i, bestV, bt, v, sum: integer;
      F', F0, F1: BGF;
begin
  bestTree := 0;
  if (d >= depth_limit)
  begin
    if (WeToPlay) return Estimate(Lo(BGF));
    else return Estimate(Ro(BGF));
  end;

  Let F = F0 U F1 where F0 is the set of numbers in F and F1 is the set of
  non-number BGTs in F.
  if (F0 is empty) sum := 0 else sum :=  $\Sigma$  F0;
  if (F1 is empty) return sum;
  if (WeToPlay)
  begin
    bestV :=  $\alpha$  - sum;
    for i := 1 to |F1| do
    begin
      F' := replace Ti by TiL in F1;
      v := Metasearch(F', false, bt, sum+bestV,  $\beta$ );
      if (v > bestV) begin bestV := v; bestTree := i end;
    end
  end
  else
  begin
    bestV :=  $\beta$  - sum;
    for i := 1 to |F1| do
    begin
      F' := replace Ti by TiR in F1;
      v := Metasearch(F', true, bt,  $\alpha$ , sum+bestV);
      if (v < bestV) begin bestV := v; bestTree := i end;
    end
  end
  return sum+bestV;
end

```

This procedure is initially called by

Metasearch(BGF, true, best_choice, -361, 361, 0);

We believe that the approach described in this paper has essentially reduced the problem of solving Go to the problem of solving components of Go. The qualities of the BGTs, i.e. how well they capture the best local moves, in the model-set-up is essential to the success of this BGF approach to computer Go. A good implementation of this new move decision paradigm requires at least two years effort. Experimental results will be reported in a future paper.

References

- [Berlekamp & al. 1982] E. Berlekamp, J.H. Conway, and R.K. Guy, *Winning Ways*. Academic Press, New York, 1982.
- [Berlekamp 1991] E. Berlekamp, *Introductory Overview of Mathematical Go Endgames*, *Proceedings of Symposia in Applied Mathematics Volume 43*, 1991.
- [Berlekamp & Wolfe] E. Berlekamp and D. Wolfe, *Mathematical Go: Chilling Gets the Last Point*, A K Peters, Wellesley, 1994.
- [Chen 1989] K. Chen, *Group Identification in Computer Go*, Go chapter in the book "Heuristic Programming in Artificial Intelligence" edited by D. Levy & D. Beal, pp. 195-210, Ellis Horwood, Fall 1989.
- [Chen 1990] Chen, K., *Move Decision Process of Go Intellect*, pp. 9-17, *Computer Go*, No.14, 1990.
- [Chen 1998] K. Chen, *Heuristic Search in Go Game*, *Proceedings of Joint Conference on Information Sciences '98*, vol. II, pp. 274-278, 1998.
- [Chen 2000] K. Chen, *Some Practical Techniques for Global Search in Go*, *ICGA Journal*, Vol. 23, No. 2, 67-74, June 2000.
- [Chen 2001] K. Chen, "Knowledge and Search in Computer Go", *The 6th Game Programming Workshop*, 94-101, 2001.
- [Conway] J. H. Conway, *On Numbers and Games*, Academic Press, 1976.
- [Kao 1997] K. Kao, *Sums of Hot and Tepid Combinatorial Games*, Ph.D. thesis, University of North Carolina at Charlotte, 1997.
- [Kao 2000] K. Kao, *Mean and temperature search for Go endgames*, *Information Sciences*, 122, 2000, pp. 77-90.
- [Kao & Chen 1992] K. Kao, K. Chen, *End Game Theory*, *The 4th Computer Olympiad Game Conference*, London, 1992.
- [Mueller 1995] M. Mueller, *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*, Ph.D. Dissertation, Swiss Federal Institute of Technology Zurich, 1995.
- [Mueller 1999] M. Muller, *Decomposition Search: A combinatorial games approach to game tree search, with applications to solving Go Endgames*, In *IJCAI-99*, pp. 578-583, 1999.
- [Mueller 2001] M. Muller, *Global and local game tree search*, *Information Sciences*, Vol. 135, Nos. 3-4, pp. 187-206, July 2001.
- [Nowakowski 1996] R. Nowakowski, editor, *Games of No Chance*, Cambridge University Press, 1996.

