Decomposition Search: A Combinatorial Games Approach to Game Tree Search, with Applications to Solving Go Endgames

Martin Muller
Electrotechnical Laboratory
Tsukuba, Japan
mueller@etl.go.jp

Abstract

We develop a new method called *decomposition search* for computing minimax solutions to games that can be partitioned into independent subgames. The method does not use traditional minimax search algorithms such as alpha-beta, but relies on concepts from combinatorial game theory to do locally restricted searches. This divide-and-conquer approach allows the exact solution of much larger problems than is possible with alpha-beta.

We show an application of decomposition search to the game of Go, which has been traditionally regarded as beyond the range of exact search-based solution methods. Our experiments with solving endgames show that alpha-beta searches already become impractical in positions with about 15 remaining moves. However, an endgame solver based on decomposition search can solve a much larger class of endgame problems with solution lengths exceeding 60 moves.

1 Introduction

In two-player games with perfect information, minimax-based search methods have been very successful. Games such as 4-in-a-row, gomoku or nine men's morris have been solved, and heuristic game-playing programs have reached world championship level in a number of popular games. In chess and checkers, endgame databases constructed using retrograde analysis have uncovered a wealth of new information and forced the rewriting of the textbooks.

Today, the conditions under which these standard approaches are successful are well understood. One class of games in which they have not succeeded is combinatorial games. Such a game can be represented as a combinatorial *sum* of local games, called *subgames.*

Decomposition search is a new computational method for solving combinatorial games. Decomposition search decomposes a game into a sum of subgames, performs a particular kind of local search for each subgame, applies combinatorial game theory to evaluate the resulting local game graphs, and determines overall optimal play from the combinatorial game values of subgames.

By reducing the scope of searches from global to local, the new method can compute minimax solutions and determine optimal play in such games much faster than classical techniques such as alpha-beta, which can not exploit the extra structure given by the decomposition.

The structure of this paper is as follows: Section 2 reviews several existing divide-and-conquer approaches to solving games, including the combinatorial game-theoretical approach to the analysis of games with decomposable state. Section 3 introduces *decomposition search,* a four step algorithm for finding the minimax solution and optimal play in combinatorial games. Section 4 applies decomposition search to Go endgames, and Section 5 compares the performance of decomposition search with standard alpha-beta game tree search, using Go endgames as examples.

2 Divide-and-Conquer Approaches to Solving Games

Research in game tree search follows two general goals: reducing the size of the search space, and traversing the space in clever ways in order to find solutions early. For solving games with large state spaces, divide-and-conquer approaches are attractive: the idea is to identify simpler subproblems that can be solved more easily, and can contribute to an overall solution.

2.1 Heuristic Problem Decomposition: Identifying Subgoals

In complex games, the ultimate goal of the game is difficult to reach directly. Therefore, players identify subgoals and search specifically to achieve these goals. For example, bridge players analyze single-suit play, and chess players seek ways to capture a particular piece or break through a pawn chain. A narrowly focused search to achieve a subgoal is typically much easier than full width game search, yet achieving a subgoal can have a significant impact on the outcome of the game.

2.2 Splitting a Game Vertically: Endgame Databases

A very successful divide-and-conquer method in the computational analysis of games has been the construction of endgame databases. In this approach, a game is split vertically into progressively simpler games along the time axis. *Converging* games such as checkers, nine men's morris or chess can be split in this way, because they simplify towards the end of the game when fewer and fewer pieces remain on the board.

Endgame databases are built bottom-up, starting from the simplest subgames, by retrograde analysis [Thompson, 1986]. The optimal play outcome, and optionally the distance to a win or conversion to another subgame, is computed for all positions in the subgame. Databases are used during heuristic search: whenever search hits a database position, the exact value can be used in place of the heuristic evaluation.

2.3 The Mathematics of Decomposition: The Combinatorial Game Approach to the Analysis of Games

Combinatorial game theory [Conway, 1976; Berlekamp *et* al., 1982] provides the mathematical basis for a more radical divide-and-conquer method: it breaks up game positions into pieces and analyzes the overall game in terms of these smaller local subgames.

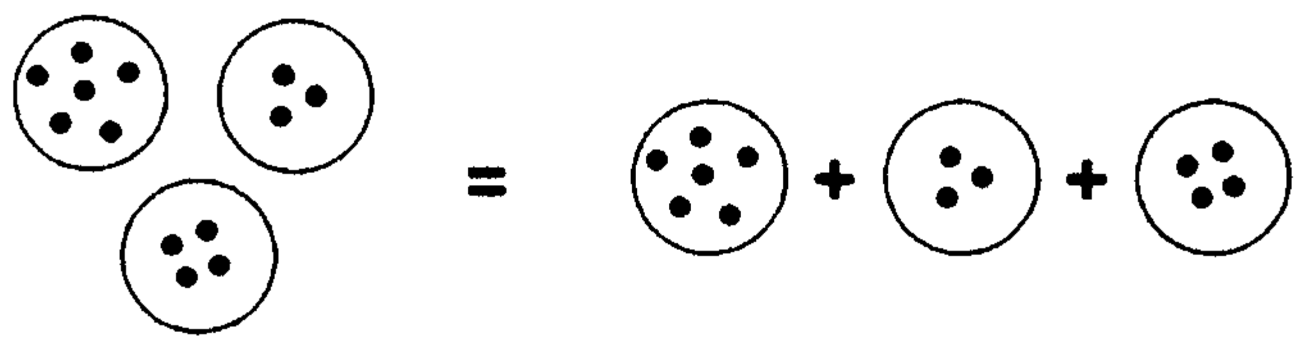
Figure 1: A three heap *Nim* position and its subgames

Each move in the game corresponds to a move in one subgame and leaves all other subgames unchanged. A game ends when all subgames have ended, and the final outcome of the game can be determined from the subgame outcomes. A well-known example of a combinatorial game is *Nim,* shown in Figure 1, which is played with heaps of tokens. At each move, a player removes an arbitrary number of tokens from a single heap, and whoever runs out of moves first loses. Each Nim heap constitutes one subgame. While winning a single subgame is trivial, winning the *sum* of several heaps requires either exhaustive analysis, or, much more efficiently, a computation using the calculus of combinatorial games.

3 Decomposition Search

This section develops decomposition search as a framework for solving games through decomposition, a particular kind of local search named *local combinatorial game search (LCGS)* and the analysis of the resulting local game graphs by applying combinatorial game theory.

3.1 Definition of Decomposition Search

Let *G* be a game that decomposes into a sum of subgames $G_1 + \ldots + G_n$. Let the combinatorial game evaluation of *G* be $C(G)$. *Decomposition search* is defined as the following four step algorithm for determining optimal play of G:

1. Game decomposition and subgame identification: given *G,* find an equivalent sum of subgames $G_1 + \ldots + G_n$.
2. Local combinatorial game search (LCGS): for each G,, perform a search to find its game graph $GG(G_i)$
3. Evaluation: for each game graph $GG(G_i)$, evaluate all terminal positions, then find the combinatorial game evaluation of all interior nodes, leading to the computation of $C(G_i)$.
4. Sum game play: through combinatorial game analysis of the set of combinatorial games $C(G_i)$, select an optimal move in $G_1 + \ldots + G_n$.

The following subsections describe the four steps of decomposition search in more detail, discuss how to use the results of decomposition search during game play, and describe limitations of the method.

3.2 Game Decomposition and Subgame Identification

The precondition for applying decomposition search to a game position is that it can be split into subgames which fit the combinatorial games model outlined in Section 2.3. The specific decomposition procedure depends on the game rules.

In some games, such as Nim, Amazons and many of those analyzed in the book Winning Ways [Berlekamp *et* a/., 1982], a suitable decomposition follows directly from the rules of the game. Figure 1 shows the decomposition of a Nim position. In other games, such as Go, more game-specific knowledge is necessary to find a good decomposition of a given position.

3.3 Local Combinatorial Game Search

Local combinatorial game search (LCGS) is the main information gathering step of decomposition search. It is performed independently for each subgame. LCGS generates a game graph representing all relevant move sequences that might be played locally in the course of a game. LCGS works differently from minimax tree search in a number of ways, including move generation and recognition of terminal positions.

Differences Between LCGS and Minimax Search

The game graph built by LCGS differs from the tree generated by minimax search. In the case of minimax, players move alternately, so each position is analyzed with respect to the player on move. In contrast, there is no player-to-move-next in a subgame. All possible local move sequences must be included in the analysis, including sequences with several successive moves by the same player, because players can switch between subgames at every move.

Another difference between local and full state search is the treatment of cycles. To prevent infinite games, the repetition of a game position is forbidden in most games, or limited to a small number as in chess. However, the same local position can re-occur repeatedly as long as the whole game keeps changing. Combinatorial game evaluation is defined only for games without cycles. Therefore, decomposition search deals only with locally acyclic games, and with those cyclic games where cycles do not enter into optimal play.

Move Generation

LCGS must generate all legal local moves for both players, except in a terminal position or if moves can provably be pruned. Such exact pruning rules are game-specific. Examples are restricting the number of equivalent moves generated to a single one, or pruning locally bad moves which are dominated by other moves.

Terminal Positions and Local Scoring

Termination rules decide when a position can be evaluated without further expanding the game graph. LCGS defines the following termination rules:

- No legal moves
- No good move, game recognized as constant
- Value of position already known

The first two cases represent local terminal positions, which evaluate to an integer. This number, the *local score,* is game-specific and computed according to the rules of the game.

In the third case, if the value of a position is already known from another source, such as a transposition table, game-specific knowledge, or a precomputed local position database, LCGS can be terminated as well. The value retrieved for such a position is a combinatorial game, which has previously been computed by local evaluation as discussed in the next section.

3.4 Local Evaluation: Mapping Game Graphs to Combinatorial Games

Local evaluation computes the combinatorial game value of a given acyclic local game graph with evaluated leaf nodes. Let the players be Black and White, with positive scores good for Black. If from a local position *p* Black can move to $b_1 \dots b_n$ and White can move to $w_1 \dots w_m$, and if the evaluations of these follow-up positions are already known, then the evaluation *C(p)* is given by the combinatorial game expression

$$C(p) = \{ C(b_1), \dots, C(b_n) \mid C(w_1), \dots, C(w_m) \}.$$

This expression can be brought into a canonical form using standard rules of combinatorial game theory. Repeated bottom-up application of the formula eventually yields an evaluation of each node in the game graph.

Cycles that do not Affect the Game Value

Cycles can occur during LCGS, even if they have no effect on optimal play. If evaluation fails due to cycles, bounds are computed by forbidding one player all moves that would repeat a position. This transforms the game graph into an acyclic graph, a different one for each player. If both bounds coincide, optimal play does not depend on cycles. Otherwise, decomposition search stops and indicates a local evaluation failure.

3.5 Sum Game Play

To find an optimal move in a sum game, the final step of decomposition search selects a move which most improves the position. This improvement is measured by a combinatorial game called the *incentive* of a move. The incentives of all moves in all subgames are computed locally. If one incentive dominates all others, an optimal move has been determined. This is the usual case for games with a rich set of values such as Go.

Since incentives are combinatorial games and therefore only partially ordered, it can happen that more than one nondominated candidate move remains. In this case, an optimal move is found by a more complex procedure involving the combinatorial summation of games [Conway, 1976].

Since such a summation can be an expensive operation, there is no worst case guarantee that decomposition search is always more efficient than minim ax search. In practice, it seems to work much better. The algorithm presents many opportunities for complexity reduction of intermediate expressions during local evaluation as well as during summation.

Even though all search and most analysis is local, decomposition search yields globally optimal play, which can switch back and forth between subgames in very subtle ways, as in the example of Figure 8.

3.6 Reusing Decomposition Search Results During Play

The result of decomposition search is a complete description and evaluation of all reasonable local play sequences, which makes perfect overall play possible. Results of local analysis can be saved in a database. During play, each full board position corresponds to a set of matching local positions, one from each subgame. Positions and their combinatorial game values are retrieved from the database.

As long as the opponent follows analyzed lines, followup moves can be played from the information stored in the database, without further search. If the opponent plays a less-than-optimal move that was pruned during LCGS and reaches an unevaluated position, the corresponding subgame is re-searched from the new position.

3.7 Limitations of Decomposition Search

There are two types of limitations for decomposition search: cyclic subgames and bounded computational resources. As discussed in Section 3.4, cyclic subgames can be handled only in the case where cycles don't affect optimal play.

Resource exhaustion is detected during algorithm execution if any of the following hold: game decomposition fails or results in very large subgames, LCGS exceeds a preset time or space limit, or intermediate combinatorial game expressions become too complex. Practical limits are highly game-specific, and depend on the shape of local game graphs built during LCGS and on the complexity of the combinatorial games involved. For example, *impartial* games such as Nim are generally easier to evaluate than *partizan* games such as Go.

4 Applying Decomposition Search to Go Endgames

This section discusses how to apply decomposition search to the game of Go. Game decomposition is achieved through the recognition of safe stones and territories and the resulting board partition. Other Go-specific aspects are pruning moves during LCGS and scoring of terminal positions.

4.1 Subgame Identification in Go by Board Partition

A Go position can be decomposed when parts of the board are isolated from the rest by walls of safe stones. Moves in one part have no effect on other parts across such a wall. Figures 2 and 3 show the two decomposition steps: first, finding safe stones and territories, and then identifying subgames as the connected components of the remaining points on the Go board.

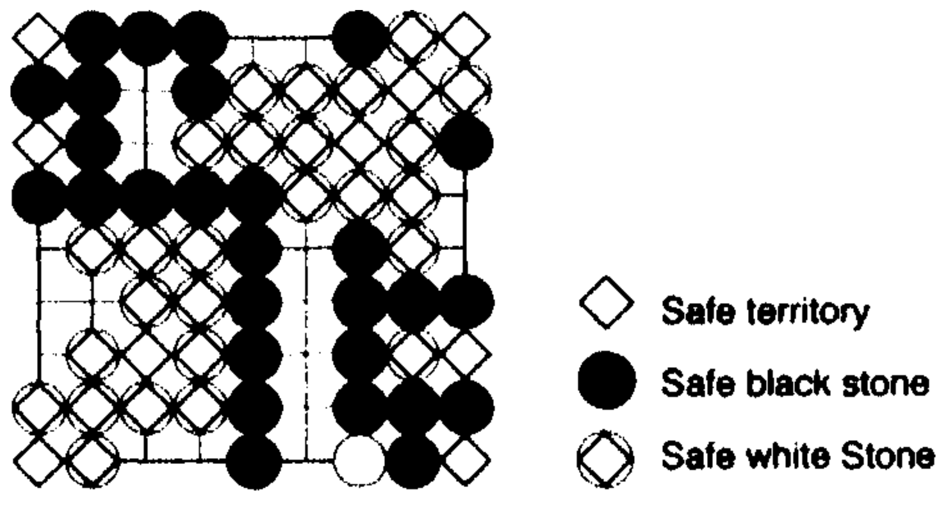

Figure 2: Recognition of safe stones and territories

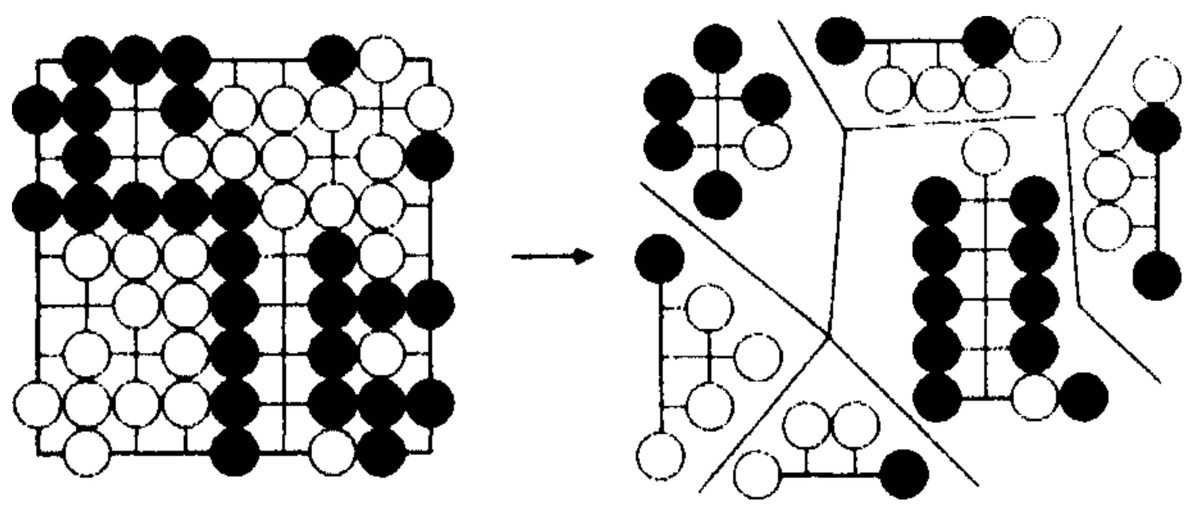

Figure 3: Decomposition of Go endgame position

Finding Safe Stones and Territories

Safe territories are 'finished' subgames: they can be evaluated by a number, the size of the territory. Areas which are completely surrounded by one player are candidate territories. Territories are found by goal-directed search, applying the techniques of [Miiller, 1995; 1997] to prove the safety of candidate territories.

Play in territories that have been proven safe is simple. The player never plays first in any territory. If the opponent attacks the player's territory, a goal-directed search is performed to find a refutation which restores the safety of the area.

4.2 LCGS in Go

An endgame area consists of unsettled stones, and of empty points which are not territory. Safe stones, usually of both colors, surround each endgame area, as shown in Figure 3. During endgame play, unsettled stones either become safe or are captured. Empty points will either be occupied or become part of a safe territory. A rare special case are shared empty points in *seki.*

Scoring Local Terminal Positions in Go

Scoring assigns an integer to each terminal position. In Chinese rules, scoring measures the difference between how many stones and empty points belong to either color. In Japanese rules, territory and prisoners are counted. Both kinds of scoring are straightforward in a terminal position since the status of all stones and empty points is known exactly.

Pruning Moves

In contrast to the speculative pruning in selective search methods, only moves that are provably worse-or-equal than others can be eliminated. For example, if a move achieves control of all points in the local area, it is optimal, and all other moves can be pruned. In almost surrounded areas such as the one shown in Figure 4, the move at the entrance at *a* is the only good move for either player.

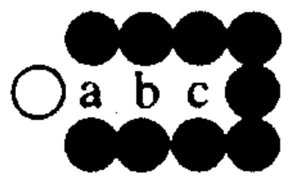

Figure 4: Area with unique best move at *a*

4.3 Full Board Move Selection in Go

Full board move selection in Go distinguishes three cases:

1. If the opponent just made a threat in player's territory, reply as in Section 4.1 to keep territory and stones safe.
2. Otherwise, if the combinatorial game is not finished yet, play the sum game as in Section 3.5.
3. Otherwise, perform a cleanup phase: fill in the final neutral points to finish the game.

5 Experiments

The performance of decomposition search is compared with standard full board alpha-beta search on two representative examples from a set of Go endgame puzzles in [Berlekamp and Wolfe, 1994]. In the examples, territories have been slightly strengthened to make it easier to prove their safety. The endgames are equivalent to the original version. In each experiment, the full board problem was solved from scratch. No precomputed database of subgames was used.

5.1 Full-board Minimax Search

The minimax implementation used a standard alpha-beta search. The size of the transposition table was 32k entries for the small problems, 4M entries for the big ones. Since naive full-board search would be too expensive, alpha-beta search was allowed to use the same knowledge about safe territories and the same local pruning rules as LOGS.

In contrast to LCGS, pass moves must be generated, because in positions where there is no good move, players must be allowed to pass, instead of being forced to damage their own position.

5.2 First Example: 20 Point Problem

The first Go endgame example, on a 9 x 9 board, is based on problem C.9 of [Berlekamp and Wolfe, 1994]. After computing safe stones and territories, the total remaining endgame area is 20 points. There are six regions labeled A to F, with sizes ranging from 2 to 6.

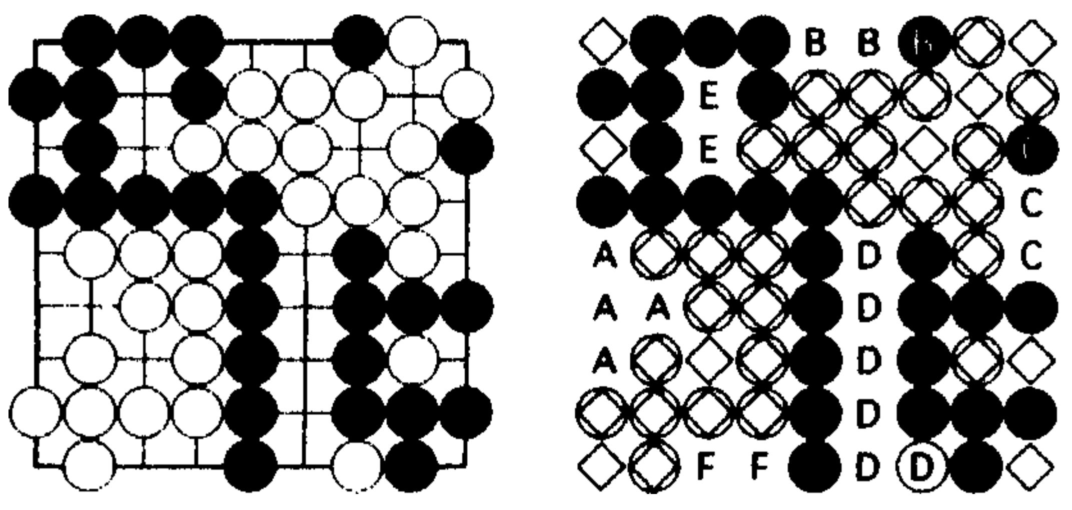

Figure 5: Problem C.9 and its decomposition

This problem is trivial for decomposition search, yet already challenging for minimax. For more detailed testing, a series of simplified problems was created, in which several local endgame situations were replaced by constant territories. Figure 6 shows such a simplified problem of size 10 consisting of areas A, B and C. Areas D, E and F have been 'played out' and replaced by constant territories, as shown by the markings in the figure.

White is to play first in all problems. Table 1 shows the total node count for the LCGS phase of decomposition search, followed by the node count and solution time in seconds for alpha-beta, as measured on a Macintosh G3/250. The solution times for decomposition search are not shown, since they were all very similar at 0.2 - 0.3 seconds. The size of the transposition table (4M entries) was insufficient for the full 20 point problem, resulting in an enormous increase in solution time to over 28 hours.

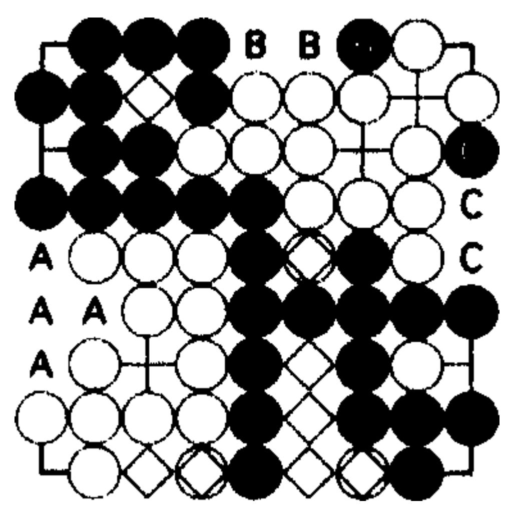

Figure 6: Problem C.9 reduced to areas A + B + C

Areas (Size)	Nodes DS	Node* aB	Time aB
A (4)	21	39	<0.1
A + B (7)	26	526	0 1
A + B + C (10)	31	5905	1 9
A + B + C + D (16)	42	1097589	295 9
A + B + O + D + E (18)	45	10243613	2461.0
A + B + C + D + E + F (20)	48	463941123	103406.2

Table 1: Comparison of decomposition search and alpha-beta in problem C.9

5.3 Second Example: 89 Point Problem

The second example, C.II of [Berlekamp and Wolfe, 1994], is a Go endgame problem on a 19 x 19 board. Figure 7 shows the initial position and its partition into subgames. After determining safe stones and territories, 89 unsettled points remain, partitioned into 29 distinct endgame areas of sizes 1 to 6.

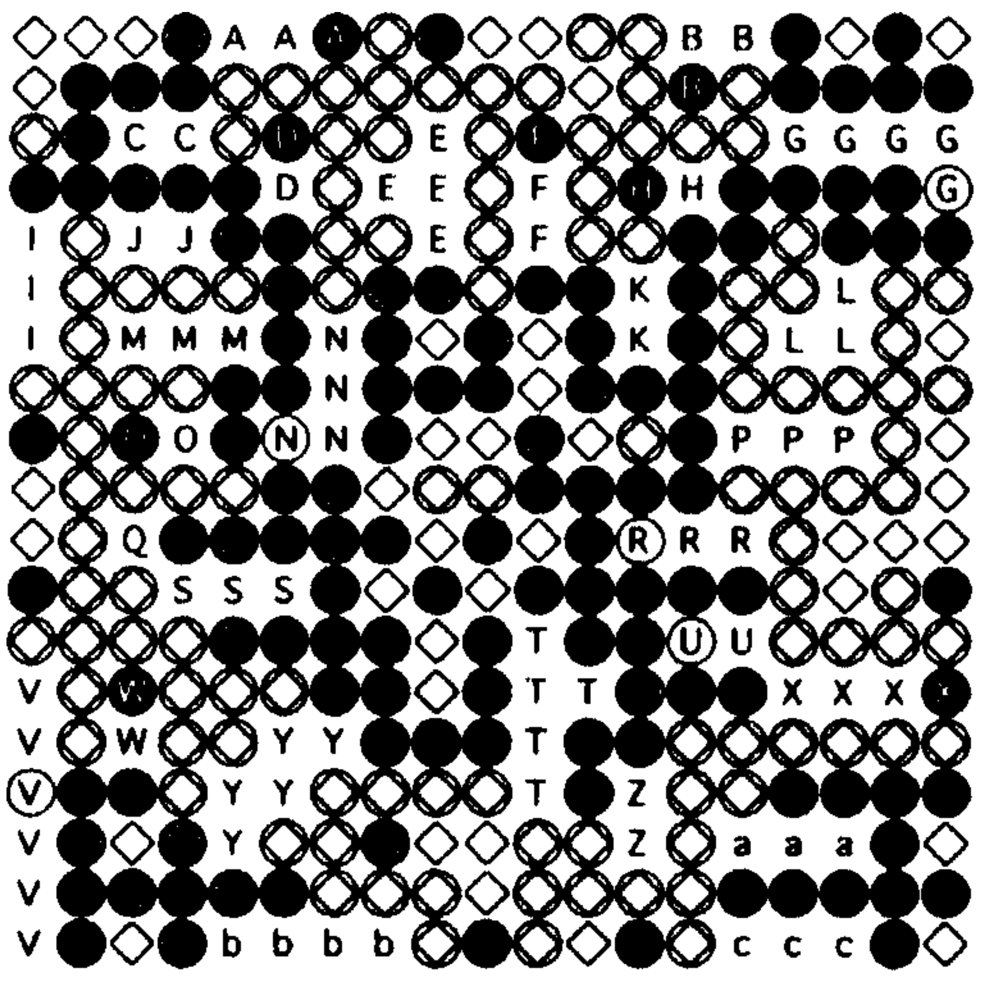

Figure 7: C.II: an 89 point endgame problem

An optimal 62 move solution sequence computed by decomposition search is shown in Figure 8. On the system described above, the complete solution takes 1.1 seconds, including 0.4 seconds for LCGS searching a total of 420 nodes in the 29 subgames. The remaining time is taken up by proving the safety of territories and by operations on combinatorial games. Alpha-beta search behaved as in the first example. Node counts and solution times for the first few subproblems are shown in Table 2.

Areas (Size)	Node* DS	Node* *aB*	Time aB
A (3)	5	15	<0.1
A + B (6)	10	178	<0 1
A + B + C (8)	13	899	0.3
A + B + C + D (10)	16	2663	0.7
A + B + C + D + E (14)	21	45446	16.7
A + B + C + D + E + F (17)	26	209815	66.1
A + B + C + D + E + F + G (22)	35	10350151	3192.9
A + B + C + D + E + F + G + H (24)	38	78629573	25044 2

Table 2: Performance of alpha-beta on problem C.II

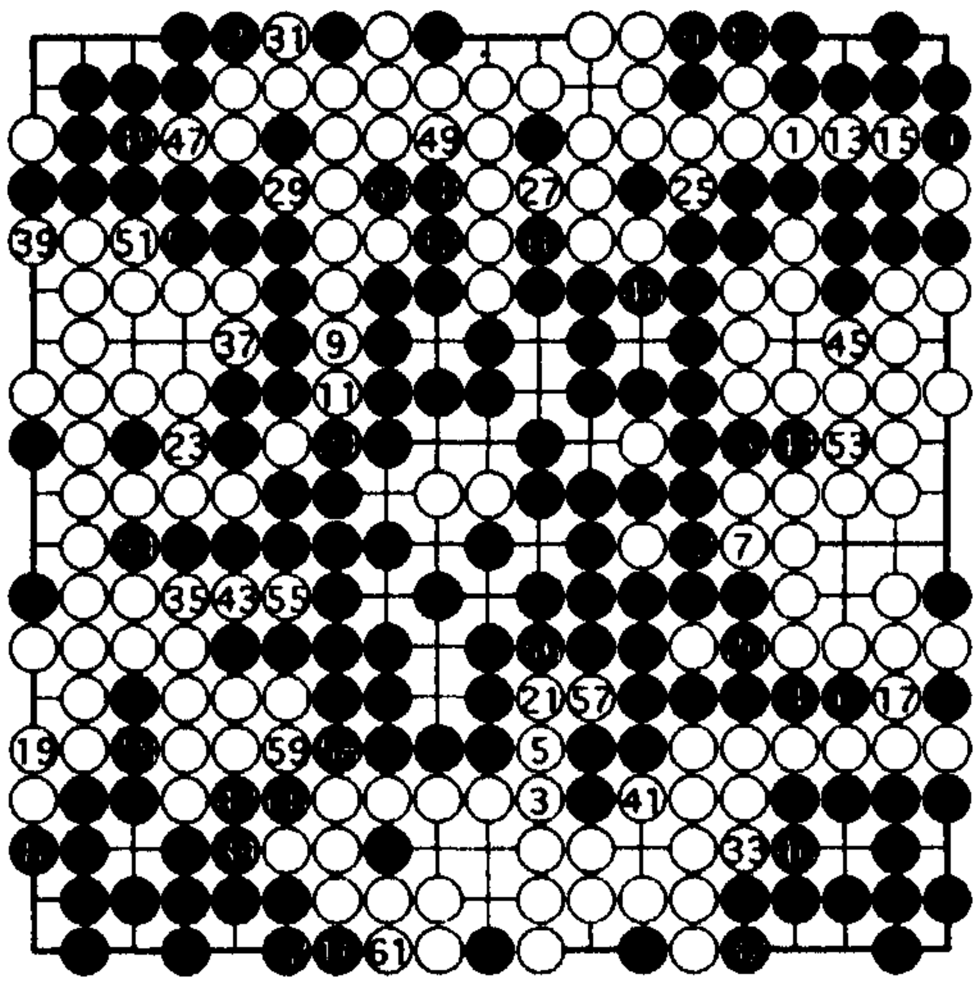

Figure 8: An optimal solution to problem C.II

As a final test two games starting from the initial position of C.II were played against the current world champion Go program *The Many Faces of Co.* Playing Black, the decomposition search program gained one point over the game-theoretically optimal result. Playing White, it gained five points. Considering the small differences in value of the endgame plays involved, the total gain of six points in two experiments is significant.

5.4 Discussion

The discussion compares decomposition search and alpha-beta in terms of time requirements, results generated and information on alternative moves.

Time Requirement

The fundamental disadvantage of alpha-beta relative to decomposition search is clearly demonstrated by the results: alpha-beta requires time that is exponential in the size of the *whole problem,* while LCGS' worst case time is exponential in the size of the *biggest subproblem.* If local combinatorial game evaluations can be computed and compared without too much overhead, a dramatic speedup results.

Reusing Partial Results

Another advantage of decomposition search over alpha-beta is that it generates useful partial results in the form of evaluated subgames. Frequently occurring games and their combinatorial game evaluation can be stored in a persistent database. If some local searches can be avoided or terminated early by a database hit, further speedups result. This method works for any combinatorial game, whereas in the case of minimax search databases can be built only for the endgame phase of converging games.

Information on Alternative Moves

Alpha-beta returns the best move and the minimax score of a position. Evaluating alternative moves requires more search. On the other hand, data generated during decomposition search easily yields further information such as other optimal moves and the amount by which a bad move is inferior to an optimal one.

6 Summary

Decomposition search is a new computational method to find minimax solutions of combinatorial games. The method provides a framework to restrict search to subgames, and uses powerful mathematical techniques of combinatorial game theory to combine the local results and achieve globally optimal play. As a divide-and-conquer method, decomposition search results in vast improvements compared to alpha-beta search.

An application of decomposition search to Go has demonstrated perfect play in long endgame problems, which far exceed the capabilities of conventional game tree search methods.

References

[Berlekamp and Wolfe, 1994] E. Berlekamp and D. Wolfe. *Mathematical Go: Chilling Gets the Last Point.* A K Peters, Wellesley, 1994.

[Berlekamp *et al,* 1982] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways.* Academic Press, London, 1982.

[Conway, 1976] J. Conway. *On Numbers and Games.* Academic Press, London/New York, 1976.

[Muller, 1995] M. Muller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory.* PhD thesis, ETH Zurich, 1995. Diss.Nr. 11.006.

[Muller, 1997] M. Muller. Playing it safe: Recognizing secure territories in Computer Go by using static rules and search. In H. Matsubara, editor, *Proceedings of the Game Programming Workshop in Japan '97,* pages 80-86, Computer Shogi Association, Tokyo, Japan, 1997.

[Thompson, 1986] K. Thompson. Retrograde analysis of certain endgames. *ICC A Journal,* 9(3): 131-139, 1986.