ELSEVIER

Information Sciences 135 (2001) 187–206

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Global and local game tree search

Martin Müller [1]

Electrotechnical Laboratory, Umezono 1-1-4, Tsukuba 305, Japan

Received 20 June 2000; received in revised form 16 January 2001; accepted 8 March 2001

Abstract

Computer programs based on minimax search have achieved great success, solving a number of classic games including Gomoku and Nine Men's Morris, and reaching a performance that approaches or surpasses the best human players in other well-known games such as checkers, Othello and chess. All these high-performance game-playing programs use *global search* methods, which evaluate complete game positions. *Local search* is an alternative approach that works well in the case where a game state can be partitioned into independent subgames. The method of *decomposition search* [M. Müller, in: IJCAI-99, 1999, p. 578] can solve such games by a combination of *local combinatorial game search* with evaluation techniques from combinatorial game theory. We compare local and global search in endgame problems in the game of Go, which has been traditionally regarded as beyond the range of exact search-based solution methods. An endgame solver based on decomposition search can solve problems with solution lengths exceeding 60 moves. © 2001 Elsevier Science Inc. All rights reserved.

Keywords: Computer Go; Local search; Game tree search; Decomposition search; Combinatorial game theory

1. Introduction

In two-player games with perfect information, minimax-based global search methods have been very successful. Games such as 4-in-a-row, Gomoku or

E-mail address: mmueller@cs.ualberta.ca (M. Müller).

[1] Present address. Department of Computing Science, University of Alberta, Edmonton, Canada T6G 2E8.

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 the standard approach is successful are well understood. One class of games in which they have not succeeded is combinatorial games [2,6]. Such games can be represented as the combinatorial *sum* of local games, called *subgames*. Combinatorial games can have a very large state space, which makes global search impractical. However, because of their special structure, other, more efficient solution methods based on local search become possible.

Decomposition search [11] is a 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, and thereby exploiting the extra structure given by the decomposition, the method can compute minimax solutions and determine optimal play in such games much faster than global minimax search techniques such as alpha–beta.

The structure of this article is as follows: Section 2 discusses global and local search in games, with a focus on divide-and-conquer approaches to solving games. One such approach is using combinatorial game theory for the analysis of games with decomposable state. Section 3 describes the local search-based method of decomposition search, which consists of four steps for finding the minimax solution and optimal play in combinatorial games. Section 4 introduces Go endgames as a case study, and discusses how to apply both the global search method of minimax search and the local search method of decomposition search to this problem. Three factors that complicate minimax search in Go as compared to other games are identified. Section 5 compares the characteristics and performance of global and local search in Go endgames, and Section 6 summarizes the research and closes with a discussion of future work, with the goal of combining the efficiency of local search with the generality of global search.

2. Divide-and-conquer approaches in game tree search

A search engine performing deep global minimax searches forms the core of most computer programs for two-player games with perfect information. This search method has led to overwhelming success in many popular games, such as chess, checkers, shogi, Othello, awari, Chinese chess, Gomoku, and Nine Men's Morris. However, the same approach has not worked as well in a number of other games, especially the so-called combinatorial games. Frequent reasons cited for problems with applying global minimax search to games are: the large number of possible moves in a typical game position, the length of a game, or the difficulty of developing an accurate evaluation function. A well-known game that exhibits all these difficulties is the ancient Asian game of Go.

In science, the standard approach to dealing with great complexity is divide-and-conquer. If a problem can be decomposed into independent subproblems that can be solved more easily, an overall solution may become feasible. A number of such divide-and-conquer approaches have been used successfully in game tree search. Sections 2.1 and 2.2 review problem reduction techniques within the global search framework. The remainder of Section 2 and Section 3 describe the more radical divide-and-conquer approach of decomposition search which is based on local search.

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 overall outcome of the game. Despite the intuitive appeal of such approaches, they do not seem to be used much in real, competitive programs. On problem with this approach that apparently has not yet been overcome is to formulate sufficient constraints on a goal-directed search to avoid other losses while single-mindedly pursuing a particular goal.

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 the method of retrograde analysis [17]. The optimal play outcome, and optionally the distance to a win or conversion to a simpler subgame, is computed for all positions in the subgame. This method is still a kind of global search, since it always evaluates complete game positions. However, in such endgames the total number of states in the search space is greatly reduced because of the strict limitations on the number and type of pieces allowed.

Databases are used during heuristic search of the full game: 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 [2,6] provides the mathematical basis for a more radical divide-and-conquer method, which allows the use of local search in place of global search. The combinatorial game approach is to break up game positions into smaller pieces, and then analyze the overall game in terms of these local subgames. In the combinatorial game view, each move in the overall game corresponds to a move in exactly one subgame, and leaves all other subgames unchanged. A game ends when all subgames have ended, and when this happens the final outcome of the game can be determined in a simple way from the subgame outcomes, for example by adding up local scores as in Go.

A well-known example of a combinatorial game is *Nim*, shown in Fig. 1, which is played with heaps of tokens. At each move, a player removes an arbitrary number of tokens from a single heap. The first player who cannot make a move anymore loses the game. In terms of combinatorial game theory, each Nim heap constitutes one subgame. The global ending condition is very easy to check. A game is over only if all tokens on all heaps have been taken.

Nim is surely one of the simplest interesting games, and the interesting play results from its combinatorial game structure. Winning a single subgame is trivial and can be accomplished by removing all tokens of a heap at once, which leaves the opponent without a move locally. However, winning the *sum* of several heaps requires a subtle strategy, which can be determined either by exhaustive analysis, or, much more efficiently, by a short computation using the calculus of combinatorial games. A well-known equivalent method of computing Nim values is "binary XOR addition".

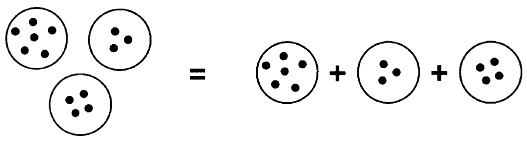

Fig. 1. A three heap *Nim* position and its subgames.

2.4. Combinatorial game methods in computer game-playing

Early ad hoc programs for solving some combinatorial games, especially *impartial* games where both players have the same move options, date back more than 25 years. These programs were used to find regularities and empirically test mathematical hypotheses in Nim-like games. The results are summarized in the book "Winning Ways" [2].

As part of his Ph.D. research about ten years ago, David Wolfe developed a software toolkit for combinatorial games, which is freely available over the Internet [18]. This toolkit contains an efficient implementation of a data structure for combinatorial games, and provides a great variety of operations on such games. The toolkit has been used as the engine for a number of computer implementations of combinatorial games. The collection "Games of No Chance" [15] contains descriptions of several such programs. Wolfe's toolkit is also used in our program for solving Go endgames [9], the first game-playing program that applies combinatorial game theory in a game with a nontrivial decomposition. Based on this prior practical experience, [11] develops *decomposition search* as a general framework for solving games by decomposition, followed by a particular kind of local search named local combinatorial game search (LCGS), and the analysis of the resulting local game graphs through combinatorial game theory. Since the method is not as well known as global minimax search, we describe decomposition search in some detail in Section 3, before comparing the two approaches in Sections 4 and 5.

3. Decomposition search

Decomposition search is a general framework for solving combinatorial games by computer analysis. This section gives a concise definition of decomposition search, followed by more detailed descriptions of the four steps constituting the method, a discussion of how to use the results of decomposition search during game play, and the advantages and limitations of the method. In order to give a broad overview first, a detailed discussion of the necessary conditions that games must obey in order to use decomposition search is also deferred to later sections.

3.1. Brief definition of decomposition search

Let G be a game that decomposes into a sum of subgames $G_1 + \cdots + G_n$. 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 $G_1 + \cdots + G_n$.

2. LCGS: for each G_i, 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 $C(G_i)$ as the evaluation of the root node.
4. Sum game play: through combinatorial game analysis of the set of combinatorial games $C(G_i)$, select an optimal move in $G_1 + \cdots + G_n$ and therefore in G.

3.2. Step 1: game decomposition and subgame identification

The precondition for applying decomposition search to a game position is that it can be split into independent subgames in a way which fits the combinatorial games model outlined in Section 2.3. The decomposition procedure is game-specific.

In some games, such as Nim and many of those analyzed in the book "Winning Ways" [2], a suitable decomposition follows immediately from the rules. For example, Fig. 1 shows the decomposition of a Nim position into three subgames with one heap each. Each heap is independent from the other heaps, and it is clear from the rules that a single heap cannot be further divided into a sum of independent subgames. Other decompositions which group several heaps together in one subgame would be possible, but would lead to subgames that are larger and therefore harder to analyze.

In games such as Go and Amazons, more game-specific knowledge is necessary to find a good decomposition of a given position. Different decompositions may result depending on the amount of search used and the quality of game-specific analysis performed. For example, decomposition in Go depends on recognizing safe stones and territories, which can be a very hard problem by itself [10].

3.3. Step 2: local combinatorial game search

LCGS is the main information-gathering step of decomposition search. A search is performed independently for each subgame. LCGS generates a game graph representing all relevant local move sequences that might be played in the course of a game. As a local search method that is just one part of a global decision procedure, LCGS works differently from global minimax tree search in a number of ways.

3.3.1. Differences between LCGS and minimax search

The game graph built by LCGS differs from the tree generated by minimax search. The root node of the game graph corresponds only to the local game position, not to the full game state as in global search. The most important difference is with regard to the moves generated in each local situation: in the case of minimax, players move alternately, so each position is analyzed with respect to the player to move. In contrast, there is no player-to-move-next in a subgame of a combinatorial game. All possible local move sequences must be included in the analysis, including sequences with several successive moves by the same player, because players are free to switch between subgames at every turn.

Another difference between local and full-state search regards the treatment of cycles. To prevent infinite games, the repetition of a complete game position is forbidden in most games, or is limited to a small number of repetitions as in chess. However, in some games the same *local* position can re-occur repeatedly, as long as the whole game keeps changing. Combinatorial game evaluation is problematic in games with cycles, and is the topic of ongoing research [1,3,8,13,16]. In its basic form, decomposition search deals only with locally acyclic games, and with those cyclic games where it can be proven that cycles do not influence optimal play [14].

3.3.2. Move generation in LCGS

Usually, LCGS must generate all legal local moves for both players. Exceptions are terminal positions which can be evaluated statically, and positions where moves can provably be pruned. Such exact pruning rules are game-specific [12]. One example of a game-specific pruning rule is in the case where equivalence classes of moves can be defined. In this case the number of equivalent moves generated can be restricted to a single one in each class. Another example is pruning locally bad moves which are dominated by other moves.

3.3.3. Terminal positions and local scoring

Termination rules decide whether a position can be evaluated statically, without further expanding the game graph. LCGS defines the following general 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. The case of no legal moves is the classical ending condition for combinatorial games, identified with a game value of 0. In the second case, if the game is recognized as constant, an exact game-specific *local score* can be computed statically, according to the rules of the game. In many games this score will be an integer value, a Nim value [2], or a combination of these.

In the third case, if the value of a position is already known from another source, such as a transposition table entry, some game-specific knowledge, or a precomputed database of local positions, search can be stopped at this point as well. The known value retrieved for such a position can be any combinatorial game.

3.4. Step 3: local evaluation, or mapping game graphs to combinatorial games

Given an acyclic local game graph with evaluated leaf nodes, local evaluation computes the combinatorial game value of the root node, which corresponds to the current local situation. The evaluation process works bottom-up as follows: let the players be Black and White, with positive scores good for Black. If from a local position p Black can move to $b_1, \ldots, b_n$ and White can move to $w_1, \ldots, w_m$, and if the evaluations of these follow-up positions are already known, then the evaluation $C(p)$ is the combinatorial game consisting of black options $C(b_i)$ and white options $C(w_j)$, computed as the combinatorial game expression [2]

$$C(p) = \{C(b_1), \ldots, C(b_n) | C(w_1), \ldots, C(w_m)\}.$$

During evaluation, this expression is brought into canonical form by using standard rules of combinatorial game theory [2,6]. One beneficial effect of the transformation to canonical form is that dominated moves are pruned at this step. In a program, David Wolfe's toolkit for combinatorial games [18] can be used for this purpose. Repeated application of the evaluation formula eventually yields an evaluation of each interior node in the game graph, and finally of the root.

3.4.1. Cycles that do not affect the game value

Cycles in the game graph can occur during LCGS, and cause the recursive evaluation process to fail. However, many cycles are caused by bad moves, and therefore have no effect on optimal play. If evaluation fails due to cycles, upper and lower bounds on a game's value can be computed by first eliminating single-player loops, then restricting the move choices of one player to prevent position repetition. Such a restriction 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. This method is explained in more detail, using a Go example, in [14]. Algorithms for evaluating local games where cycles do matter are currently a subject of intense study [1,3,8,13,16].

3.5. Step 4: sum game play

To find an optimal move in a given sum of games, the final step of decomposition search selects a move which most improves the overall position for the player to move. This improvement is measured for each move by a combinatorial game called the *incentive* [2,4]. The incentives of all moves in all subgames can be 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 [6,9]. Since such a summation can be an expensive operation, there is no worst case guarantee that decomposition search is always more efficient than minimax 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 analyses are local, decomposition search leads to globally optimal play, which can switch back and forth between subgames in very subtle ways. An example will be shown in Fig. 8 of Section 5.4.

3.6. Advantages of decomposition search over global search

3.6.1. 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 cached during a game. Each full-board position corresponds to a set of matching local positions, one from each subgame. Previously analyzed positions and their combinatorial game values are retrieved from the cache.

As long as the opponent follows analyzed lines, follow-up moves can be determined purely from the stored information, without further search. On the other hand, if the opponent plays a less-than-optimal move that was pruned during LCGS and thereby reaches an unevaluated position, the corresponding subgame must be re-searched from the new position.

3.6.2. Persistent database for reusing partial results

An advantage of decomposition search over global search 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 local game database. If some local searches can be avoided or terminated early by a database hit, further speedup results. This method works for any combinatorial game, whereas in the case of global minimax search, comparable databases can be built only for the endgame phase of converging games.

3.6.3. Information on alternative moves

Efficient global minimax search methods use pruning methods such as alpha–beta. For a given position, they typically return the best move and the minimax score. Evaluating alternative moves requires further search, and does not mesh well with some of the popular search enhancements that try to determine a single best move as quickly as possible. On the other hand, by computing combinatorial game values, decomposition search easily yields further useful information, such as other optimal moves, or the amount by which a bad move is inferior to an optimal one.

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 3.4.1, cyclic subgames can currently be handled exactly only in the case where cycles do not 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 given 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 as well as on the complexity of the combinatorial games involved. For example, *impartial* games such as Nim are generally much easier to evaluate than *partizan* games such as Go, because the range of possible combinatorial game values is restricted to the small and easy-to-represent class of Nim-values.

4. A case study: comparing global and local search in Go endgames

As a case study, this section develops and compares two approaches to the problem of solving Go endgames: global search using minimax, and local search using decomposition search. The relative efficiency and other characteristics of these two approaches are compared in endgame studies similar to those composed by Berlekamp and Wolfe [4].

Go is a complex game, and a number of game-specific problems must be overcome for obtaining efficient search procedures. Some of these problems are shared by both global and local search, while others only occur in one kind of search. The following subsections discuss these issues in some detail.

4.1. Minimax tree search in Go

The objective of global search in Go is to maximize the full-board score. Because of the complex evaluation and high branching factor of Go, full-board minimax search in the opening or middle game is typically highly selective and shallow in current programs [5]. Programs search only tens or hundreds of positions per move decision, in contrast to the hundred of thousands or millions of nodes typically searched in other games. Even in tightly constrained domains such as capture search or endgames, minimax search in Go is complicated by the following three problems:

Recognizing terminal positions. In Go, it can be very difficult to judge whether a position is terminal, or whether valuable moves remain.

Pass moves. In Go, there is no *zugzwang*: in positions where there is no good move, a player is allowed to pass.

Local position repetition or ko. Full-board position repetition is illegal in Go. However, local position repetition occurs frequently.

4.1.1. Recognizing terminal positions

In many games, detecting the end of the game is simple, for example if a specific piece such as the chess king is captured, or if a player runs out of moves in Nim. In Go, a position is terminal if no more points are contested, and all points can be classified as black, white or neutral. Such a classification can be hard, as shown by the similar-looking examples in Fig. 2: in each case Black has completely surrounded an area in the lower left corner of the board, including empty points and some white stones. The example on the left is a terminal position, since Black's area is safe and the white stones are captured. However, in the example in the middle, Black needs another move to prevent a move at 'a' which would save the white stones and thereby destroy the black territory, as in the example on the right.

4.1.2. Pass moves in Go

In most board games, passing is illegal. If a player cannot move, the game is over and the outcome is determined by the rules, for example a stalemate in chess.

In Go, a pass move must always be generated during minimax search, unless it can be proven that at least one other good move exists. The reason is as follows: in a position where there is no good move, a player must be allowed to pass, instead of being forced to damage his or her own position. The right side of Fig. 2 shows such a position in which all moves of both players are bad.

Adding pass moves to a search can substantially increase the size of the search space, since it increases the branching factor by one and, more

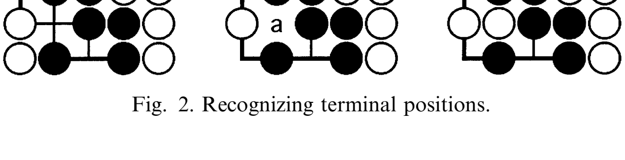

Fig. 2. Recognizing terminal positions.

importantly, leads to much longer variations containing a mixture of moves on the board and pass moves.

Local combinatorial game search on the other hand does not need pass moves, since it handles both players symmetrically in each position. Bad moves are pruned during the process of simplifying combinatorial game expressions, so players do not need an escape path in the form of passing.

4.1.3. Local position repetition by ko

Local position repetition or *ko* becomes possible if a nonlocal forcing sequence is interposed, which changes the full-board situation. Ignoring the possibility of such an event during a local search gives misleading results. In the same local position, a crucial move that is currently illegal due to position repetition might be legal after a move elsewhere on the board changes the position. The problem is that in this case local move generation is affected by a global feature, namely full board position repetition.

There is also an effect in the opposite direction, from the local situation to the analysis of the rest of the board. If the outcome of a local search depends on a player winning a ko fight, that player usually has to pay a price by ignoring some opponent moves elsewhere. In complex ko fights, many different outcomes and tradeoffs are possible. The ongoing research on local games with cycles mentioned in Section 3.4.1 is to a large degree motivated by the problem of evaluating ko fights in the game of Go.

For global search, ko is not a conceptual problem, since the game state is completely known at each time. However, ko leads to efficiency problems. Ko fights can cause a combinatorial explosion of the size of the search tree even in the case of global search, since game states which have the same board state but different move histories must be distinguished. Furthermore, different rule sets vary in the way they recognize and handle complicated repetitions with multiple concurrent ko.

4.1.4. Interactions between the three Go-specific problems

The interaction of the three Go-specific problems identified above leads to further complications:

Pass and recognition of terminal positions. In a move sequence, the number of consecutive pass moves cannot be indefinitely large. Therefore, after two or three consecutive passes the resulting position must be statically evaluated, even if the position is still unsettled. It is unclear how such an evaluation should be done.

Ko and terminal positions. Some ko fights are very favorable for one player, so in practice that player wins them by default. Some rule sets define an extra "playing out" phase with modified rules to resolve such ko. However, it seems hard to formulate general rules for handling all such cases.

Pass and ko. Pass moves made during ko fights can also lead to unresolved positions. Again, extra rules must specify the evaluation of ko when both players choose to pass.

4.1.5. Global minimax search in Go endgames

Global minimax search in Go endgame studies is a simpler search problem than the general case discussed above, since there are already many safe stones and territories, there are no ko fights, and it is easy to detect terminal positions. It can therefore provide a base line for judging the potential for global minimax search in Go.

4.2. Applying decomposition search to Go endgames

This section discusses how to apply decomposition search to endgames in Go. Game decomposition is achieved through the recognition of safe stones and territories, and the resulting partition of the rest of the board into connected components. Other Go-specific aspects in this application of decomposition search are move pruning rules in the LCGS stage, and the scoring method for terminal positions.

4.2.1. Subgame identification in Go by board partition

A Go position can be decomposed when parts of the board are isolated by a surrounding wall of safe stones. In this case, moves in one part have no effect on other parts across such a wall, with the possible exception of ko fights. Figs. 3 and 4 demonstrate the two steps of decomposition: first, finding safe stones and territories, and then identifying subgames as the connected components of the remaining points on the Go board.

4.2.2. Finding safe stones and territories

Besides providing a boundary surrounding endgame areas, safe territories in Go also contribute to the score: they can be evaluated by a number, the size of the territory. Territories and neutral *dame* points are found by goal-directed

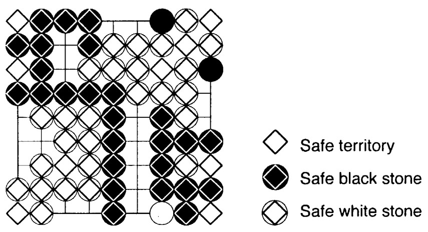

Fig. 3. Recognition of safe stones and territories.

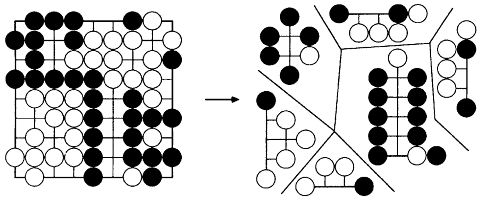

Fig. 4. Decomposition of Go endgame position.

search [9,10] which first proves the safety of candidate territories, and then recognizes points that cannot be surrounded by either player as neutral.

Play in territories that have been proven safe is simple. A competent player never plays first in any territory, neither in the own, nor in the opponent's. 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. Such a refutation must exist, since that is a logical precondition for the preceding safety proof. On the other hand, all opponent's moves in their own territory can safely be ignored.

4.2.3. LCGS in Go

A subgame in Go is an endgame area that consists of unsettled stones and of empty points which are not proven to be any player's territory. Safe stones, usually of both colors, surround each endgame area, as shown in Fig. 4. During endgame play, unsettled stones either become safe or are captured. Empty points are either occupied or become surrounded as part of a safe territory. A rare special case are shared empty points in *seki*, which are found and evaluated in the territory-proving phase.

4.2.4. 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. If the evaluation of a position is retrieved from a transposition table, its value may have to be adjusted if the same board position was reached in different ways, capturing a different number of prisoners along the way [9].

4.2.5. Pruning moves

In contrast to the speculative pruning methods used in selective search, only moves that are provably worse-or-equal than others can be eliminated. For example, if a move achieves immediate control of all the points in a local area,

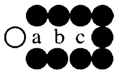

Fig. 5. Small endgame area with unique best move at *a*.

it is optimal, and all other local moves can safely be pruned. To give another example, in almost surrounded areas such as the one shown in Fig. 5, the move at the entrance at *a* is the only good move for either player.

4.2.6. Sum game play and full-board move selection

Full-board move selection in Go generally follows the model of Section 3.5, but it is complicated by the fact that the decomposition into subgames is not trivial in Go, and must be maintained in the case of opponent threats inside a safe territory. There are three cases:

1. If the opponent has just made a threat in player's territory, reply as described in Section 4.2.1 in order to keep the territory and surrounding 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. This need not be a distinct phase and can be handled by normal sum game play. However, sum game processing is more efficient if these final plays are separated from the normal, more valuable endgames.

5. Experiment: solving Go endgame studies by global and local search

The performance of decomposition search is compared with global alpha–beta minimax search on a series of Go endgames adapted from a problem in [4]. Compared to the original problems, the safe territories have been slightly strengthened to make it possible for the program to prove their safety by the methods of [9,10]. This modification does not affect endgame play, since the active endgame areas are equivalent to the original version.

5.1. Setup and scope of the experiment

The initial setup for both global and local searches in Go endgames is similar. Both methods start by finding the safe stones and territories as in Section 4.2.2. Afterwards, global search uses minimax over the whole endgame area, while local search is performed separately for each subgame. Global minimax search uses a number of standard search enhancements such as transposition tables and null window search.

Brute force global search is very time-consuming even for small problems. Especially, the pruning techniques developed for local search are not very effective in global search. These pruning techniques work best in "almost terminal" positions, which are reached much sooner in a small local search than in a global search. Therefore, a number of techniques were developed to enhance the performance of global minimax search. The following search enhancements were tested:

Move sorting (*sort*) sorts moves according to the size of the local area.

Global best move pruning (*global*) prunes all except one move candidate in positions where a globally best move exists. This is a slight generalization of the techniques developed for local search in Section 3.3.2.

Local best move pruning (*local*) prunes all other local move candidates if a locally best move is found. This method introduces local knowledge into the global search process.

Partial order move pruning (*POprune*) is an exact pruning method using a partial ordering of move classes [12] with four classes *better-than-dame*, *dame*, *pass*, and *other*. Moves are pruned in two cases: if a move in a dominating class exists, or if the move class is an equivalence class that already contains a different, equivalent move.

A global search using the local best move pruning technique can be called a *locally informed* global search. Game-specific knowledge about a local situation is used to reduce the number of candidate moves in a global search.

5.2. C.11: an 89-point Go endgame problem

Fig. 6 shows a Go endgame study equivalent to problem C.11 of [4]. In the initial position, the safe stones and territories are marked by diamonds, and the partition of the remaining points into subgames is indicated by letters. After determining safe stones and territories in this problem, 89 unsettled points remain, which can be partitioned into 29 distinct endgame areas of size 1–6. To compare the behavior of the search algorithms in more detail, a series of simplified problems was created, in which only a subset of these endgame areas must be solved.

Fig. 7 shows the number of nodes searched by both global and local search in a series of subproblems of C.11. The horizontal axis shows 13 different subsets of the endgame that were searched, from the simplest problem consisting only of the 3-point area labeled 'A' up to the union of all areas labelled 'A..M', which cover a total of 37 points, or roughly 40% of the total problem. The vertical axis shows the number of nodes searched on a logarithmic scale for the local search method LCGS and for different enhancements to alpha–beta minimax search, as discussed above.

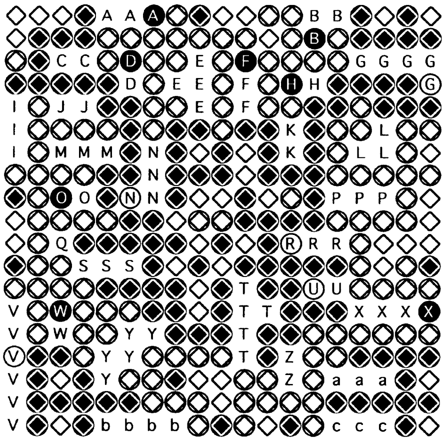

Fig. 6. C.11: an 89-point endgame problem.

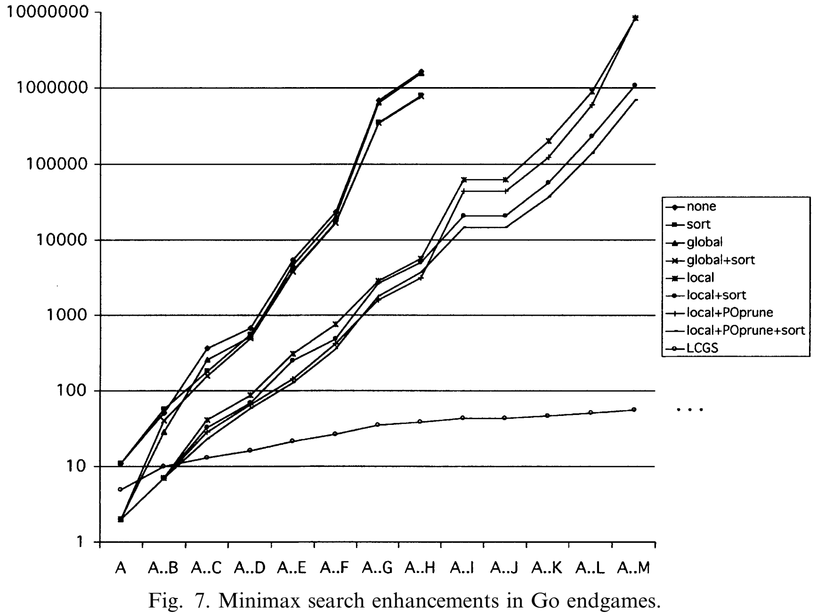

Fig. 7. Minimax search enhancements in Go endgames.

5.3. Evaluation of test results

The experiments show that global minimax search cannot compete with the local search method LCGS used in decomposition search, even with all global search enhancements in place. A solution of the full-board problem by global search seems out of the question with current methods, whereas the same problem is easily solved by decomposition search. The fundamental disadvantage of global search relative to decomposition search is clearly demonstrated by the results: global search 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 the local combinatorial game evaluations generated during LCGS can be computed and compared without too much overhead, as usually seems to be the case in these Go endgames, a dramatic speedup results.

In global minimax search, all four tested enhancements lead to substantial reductions in the number of nodes searched. The one outstanding improvement is the introduction of local move pruning, which greatly reduces both the branching factor and the depth of the search. Global and partial order move pruning are most effective in close-to-terminal positions. Earlier in the game they do not help much, since no pruning move can be found by global pruning, and almost all moves are contained in the catch-all *other* category of partial order move pruning. Move sorting works well in every combination.

5.4. Complete solution of C.11 by decomposition search

An optimal 62-move solution sequence to the full-board problem computed by decomposition search is shown in Fig. 8. On a Macintosh G3/250, the complete solution took 1.1 s, including 0.4 s for LCGS searching a total of 420 nodes in the 29 subgames. The remaining time was taken up mainly by proving the safety of territories and by operations on combinatorial games.

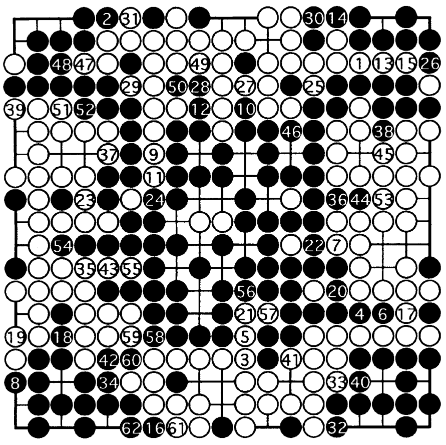

Fig. 8. An optimal solution to problem C.11.

5.5. A comparison with a commercial Go program

As a final test, two games starting from the initial position of C.11 were played against the commercially available Go program *The Many Faces of Go*, a recent world championship winner. 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 remarkable. Although these endgame problems are difficult to play perfectly, most errors cost only a fraction of a point. Even simple-minded endgame play should result in an overall loss by only one or two points. The difference of five points in the second game resulted from errors in judging the safety of territories by *The Many Faces of Go*, which made several point-losing moves inside safe territories during the game.

6. Summary and future work

Decomposition search is a new computational method to find minimax solutions of combinatorial games. The method provides a framework to restrict search to local subgames, and uses powerful mathematical techniques from 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 global 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.

Decomposition search can only be used in problems that decompose into independent subgames. In contrast, global search methods are more generally usable. Adding game-specific knowledge for local move pruning and thereby using a *locally informed* global search can greatly enhance performance and reduce the size of search trees by many orders of magnitude. Similar experiences have been reported for the single-agent search problem of Sokoban [7]. These results point the way towards developing a common framework that combines the power of local search methods with the generality of global minimax search. Such future work includes:

- A combination of local and global search in the case when there are some dependencies between local games [8].
- Integration of more local game ideas into a global search framework. For example, locally computed move incentives can be used for pruning candidate moves in a global search. Another promising idea is to exploit *miai*: if sets of two or more local games are found to cancel each other by using search, they can be removed from the search process.

- Computation of bounds on the value of nonterminal positions by means of static analysis, and use of such bounds during search.

References

[1] E. Berlekamp, The economist's view of combinatorial games, in: R. Nowakowski (Ed.), Games of No Chance, Cambridge University Press, Cambridge, MA, 1996, pp. 365–405.
[2] E. Berlekamp, J. Conway, R. Guy, Winning Ways, Academic Press, London, 1982.
[3] E. Berlekamp, Y. Kim, Where is the "$1,000 Ko"?, Go World 71 (1994) 65–80.
[4] E. Berlekamp, D. Wolfe, Mathematical Go: Chilling Gets the Last Point, A K Peters, Wellesley, 1994.
[5] K. Chen, The move decision process of Go Intellect, Computer Go 14 (1990) 9–17.
[6] J. Conway, On Numbers and Games, Academic Press, New York, 1976.
[7] A. Junghanns, J. Schaeffer, Domain-dependent single-agent search enhancements, in: IJCAI-99, 1999, pp. 570–575.
[8] Y. Kim, New values in domineering and loopy games in Go, Ph.D. thesis, University of California at Berkeley, 1995.
[9] M. Müller, Computer Go as a sum of local games: an application of combinatorial game theory, Ph.D. thesis, ETH Zürich, Diss. ETH Nr. 11.006, 1995.
[10] M. Müller. Playing it safe: recognizing secure territories in computer Go by using static rules and search, in: H. Matsubara (Ed.), Game Programming Workshop in Japan '97, Computer Shogi Association, Tokyo, Japan, 1997, pp. 80–86.
[11] M. Müller, Decomposition search: a combinatorial games approach to game tree search, with applications to solving Go endgames, in: IJCAI-99, 1999, pp. 578–583.
[12] M. Müller, Partial order bounding: using partial order evaluation in game tree search, Technical Report TR-99-12, Electrotechnical Laboratory, Tsukuba, Japan, 1999.
[13] M. Müller, E. Berlekamp, B. Spight, Generalized thermography: algorithms, implementation, and application to Go endgames, Technical Report 96-030, ICSI Berkeley, 1996.
[14] M. Müller, R. Gasser, Experiments in computer Go endgames, in: R. Nowakowski (Ed.), Games of No Chance, Cambridge University Press, Cambridge, MA, 1996, pp. 273–284.
[15] R. Nowakowski (Ed.), Games of No Chance, Cambridge University Press, Cambridge, MA, 1996.
[16] W. Spight, Extended thermography for multiple kos in Go, in: J. van den Herik, H. Iida (Eds.), Computers and Games. Proceedings of CG'98, Lecture Notes in Computer Science, vol. 1558, Springer, Berlin, 1998, pp. 232–251.
[17] K. Thompson, Retrograde analysis of certain endgames, ICCA Journal 9 (3) (1986) 131–139.
[18] D. Wolfe, The gamesman's toolkit, in: R. Nowakowski (Ed.), Games of No Chance, Cambridge University Press, Cambridge, MA, 1996, pp. 93–98.