Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

ELSEVIER

Information Sciences 154 (2003) 189–202

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Conditional combinatorial games and their application to analyzing capturing races in Go

Martin Müller

Department of Computing Science, University of Alberta, Edmonton, Canada T6G 2E8

Received 27 September 2001; accepted 24 September 2002

Abstract

Conditional combinatorial games (*CCG*) are a new tool developed for describing loosely coupled games. The definition of CCG is based on the one for classical *independent* combinatorial games. However, play in a CCG depends on its *global context*: certain moves are legal only if a nonlocal *context condition* is currently true. Compared with independent combinatorial games, CCG only allow some weaker forms of pruning. The first part of this paper starts to develop a theory of CCG and gives some examples.

In the second part, CCG are shown to be useful for studying capturing races called *semeai* in the game of Go. We introduce a general framework for analyzing semeai, which is based on CCG and on an extension called *liberty count games*. We show how this framework encompasses and extends our earlier work on solving semeai [M. Müller, Race to capture: Analyzing semeai in Go, in: Game Programming Workshop in Japan '99, IPSJ Symposium Series, vol. 99(14), 1999, p. 61].
© 2003 Elsevier Science Inc. All rights reserved.

Keywords: Combinatorial games; Conditional combinatorial games; Go; Semeai

1. Introduction

Classical combinatorial game theory [2,4] studies games that can be represented as sums of independent subgames. However, in many interesting games,

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

most notably in the game of Go, true independence of subgames is a rare event. Games can be sensibly divided into subgames which are "almost independent", but not quite. In their work on Go endgame analysis, Berlekamp and his students have frequently encountered this problem, and have invented a number of ingenious mathematical methods to deal with some cases of dependencies. In this paper, we start to develop a general framework for describing and analyzing games that can be decomposed into loosely coupled subgames. The influence of the overall game on a subgame is expressed in terms of abstract *context conditions*, which control which moves are legal in the subgame. The goal of this work is to describe weak dependencies between games in a concise form, and develop a framework that can yield a mostly local analysis of a game while taking the global dependencies between subgames into account.

The structure of this paper is as follows: in Section 2 we review some basic concepts of combinatorial game theory, then develop our new framework of *conditional combinatorial games* (*CCG*) in Section 3. In Section 4 we apply CCG to the game of Go by introducing the idea of *liberty count games* (*LCG*). We apply the new framework to capturing races in Go in Section 5, briefly survey related work in Section 6 and summarize our contributions in Section 7.

2. Combinatorial game theory

Combinatorial game theory [2,4] provides a mathematical basis for a divide-and-conquer analysis of games: it breaks up game positions into pieces and analyzes an overall game in terms of these smaller local 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 Fig. 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

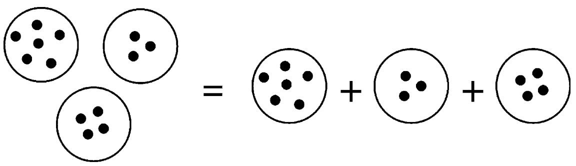


Fig. 1. A three heap *Nim* position and its subgames.

analysis, or, much more efficiently, a computation using the calculus of combinatorial games.

A combinatorial game is defined recursively by $G = \{G^{\mathrm{L}} | G^{\mathrm{R}}\}$, where G^{L} and G^{R} are sets of combinatorial games representing the move options for the players Left (L) and Right (R) in G.

3. Conditional combinatorial games

The subgames of a combinatorial game must be strictly independent from each other. In games such as Go, a strict independence rarely holds. However, human Go players routinely analyze local situations and combine the results to achieve a full-board analysis. In their work on understanding professional Go endgames, Berlekamp and his co-workers often found dependencies between subgames which had to be handled by special-purpose analyses. The idea of CCG is to develop a framework for describing loosely coupled games, which takes the dependencies between games into account, but limits the combinatorial explosion encountered in a brute force combination method such as global minimax search.

Only a restricted type of dependency between CCG is allowed, as follows: A subgame G in a CCG is similar to a subgame in classical combinatorial games. However, the legality of a move from G to an option H may depend on some abstract *context condition* C, a boolean predicate defined over the global game state S that can be evaluated at runtime. In this case, the conditional move option from G to H is written as H_C. A move H_C from G to H is legal in a global game state S if and only if C evaluates to true in S. A classical combinatorial game can be viewed as a special kind of CCG, in which none of the moves depends on a context condition, or equivalently, a CCG where all context conditions are the constant function *true*.

3.1. Example: Prime-Nim

To illustrate the idea of CCG, we introduce the game of *Prime-Nim*, with the following rules:

1. As in normal Nim, players remove a number of tokens from a single heap at each move.
2. Additionally, after each move the total number of tokens in the whole game must be a prime number.

The legal moves in a heap in Prime-Nim clearly depend on the state of all heaps. For example, in the 12 pebble starting position of Fig. 1, the legal moves are to move to a total of 11 pebbles by removing any single pebble, or to move to 7 pebbles my clearing the 5 pebble heap. All other normal Nim moves would violate the global prime number context condition. Let $*n$ denote a heap of n

pebbles, and let P be the predicate that tests that the global sum of all pebbles is prime. Then the conditional moves in Prime-Nim are as follows:

$$*n = \{*0_P, *1_P, \ldots, *(n-1)_P | *0_P, *1_P, \ldots, *(n-1)_P\}$$

For example, consider the CCG sum $*5 + *3 + *4$ of Fig. 1. The moves of each player in the subgame $*3$ are $\{*2_{\mathrm{Prime}(5+2+4)}, *1_{\mathrm{Prime}(5+1+4)}, *0_{\mathrm{Prime}(5+0+4)}\}$, which simplifies to $\{*2_{\mathrm{true}}, *1_{\mathrm{false}}, *0_{\mathrm{false}}\}$ and further to $\{*2\}$. In another sum, such as $*3 + *6$, the set of legal moves from $*3$ would be different:

$$\{*2_{\mathrm{Prime}(2+6)}, *1_{\mathrm{Prime}(1+6)}, *0_{\mathrm{Prime}(0+6)}\} = \{*2_{\mathrm{false}}, *1_{\mathrm{true}}, *0_{\mathrm{false}}\} = \{*1\}.$$

3.2. Isolated and embedded views of a CCG

A single CCG can be viewed as an abstract entity by itself. However, to relate it to a real game, the embedded view of a CCG as part of a global game is more useful.

In the isolated view, a CCG is a function that maps tuples consisting of a local game state and the boolean values of all context conditions occurring in the game to the set of successor states that are legal for the given combination of local game state and context condition values. The isolated view completely describes the behavior of a CCG under all possible combinations of context condition values.

In the embedded view of a CCG, the current value of all context conditions and therefore the set of currently legal moves are defined at each move of the global game as functions of the global game state.

3.3. Differences between CCG and combinatorial games

A very important point to note is that while we have chosen a notation that looks similar to classical combinatorial game theory, the rules for simplifying combinatorial games usually do not apply to CCG because the context conditions fundamentally alter the way that a sum of such games can be played. Each CCG is played in the context of a sum of other CCG, which are part of the overall game position. As play progresses, the truth status of context conditions can change during play, so the set of legal moves in a single CCG can change even if that subgame itself does not change.

Most properties of independent combinatorial games are lost in CCG. There is no group structure, the inverse of a game does not exist, and decomposability is lost: for a CCG in isolation, without the other games that constitute its context, not even the legal moves can be determined.

A few properties are still valid even in CCG, such as commutativity and associativity. 0 is still a neutral element, so $G + 0 = G$ for any CCG G. However, some games that simplify to 0 as combinatorial games may not simplify as

CCG because they provide context for other games. An example are ko threats in Go.

3.3.1. Example: cyclic CCG

In games such as Go, a subgame can contain cycles even if position repetition in the global game is forbidden. Fig. 2 shows an example in Go called *ko*. A simple ko position such as $A = \{1|B\}$, $B = \{A|0\}$ can be described by a CCG by indicating the context condition K in which the move from A to B is legal, and the context condition K' in which the move from B to A is legal, as follows: $A = \{1|B_K\}$, $B = \{A_{K'}|0\}$.

3.4. Pruning moves with dominated context condition in a CCG

We will call a context condition C_1 more specific than context condition C_2 if C_1 logically implies C_2. The empty context condition, which is always true, is the least specific context condition.

If one move requires a more specific context condition than another, but leads to the same CCG, it is dominated. If B_C and B_D are both options in a game and context condition D is more specific than C, then move $B_C \geqslant B_D$.

Fig. 3 shows an example in Go. Assume that the purpose of the game is to occupy all local liberties of the three connected white stones. Taking the liberty at 'A' first depends on the ko context condition, whereas taking another liberty

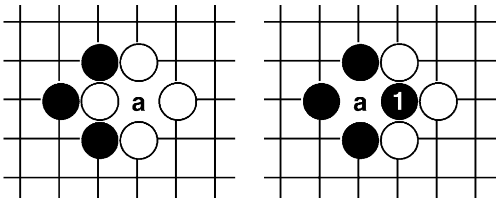


Fig. 2. Ko.

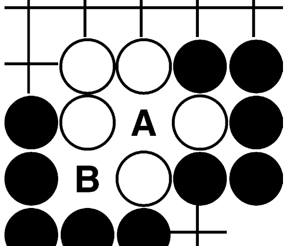


Fig. 3. Pruning a move with dominated context condition.

at '*B*' first is always legal and just as good. Otherwise, both moves are equivalent. The move *B* dominates *A* because it has an empty context condition.

4. Application to Go

Go is a "mostly local" game, and was the prime motivation for developing CCG. We study some examples of context conditions in Go, and develop the notion of liberty count games (LCG) in order to study capturing races which occur as subgames of Go.

4.1. Examples of context conditions in Go

An important context condition for local analysis in the game of Go deals with the question whether blocks of stones on the boundary of a local region can be captured. This depends on a global context condition, which we will call $L0$, that indicates whether a specific block has 0 liberties on the rest of the board. For example, in Fig. 4 the local game for a one point eye of a single white block can be described by the following CCG: $E1 = \{0_{L0}|\}$. This can be read as follows: Black can move to 0 if and only if condition $L0$ is true, that is if White does not have any liberties elsewhere. (We assume the system knows that White does not have a sensible move, so we left the set of right options empty. We also assume that capturing the whole white block ends the local game.)

4.2. Liberty count games

Like classical combinatorial games, CCG exactly describe the set of legal moves. However, they do not contain enough information to determine the number of liberties of blocks. This number is needed in order to compute the $L0$-context conditions, if one part of a Go position is used as context of another. Therefore we introduce LCG, which keep such information in addition to legal move information.

A liberty count game is defined over a set of blocks in a region and consists of two parts: a CCG that describes the possible moves of each player, and a

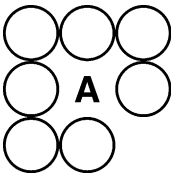


Fig. 4. $L0$-context condition of a local game.

liberty counting function $L(b, g)$ that returns the number of liberties of block b in a CCG g. A liberty counting function does not have to be defined for all blocks of a region. In semeai, typically this function will be defined only for the subset of *essential blocks* [7]. The remaining blocks are considered nonessential and not considered directly in the model, for example the stones inside a nakade shape that can be freely sacrificed. However, even those blocks are implicitly included in the model, because they affect the liberty count of essential blocks as well as the set of legal moves.

In general, the set of blocks involved in a region can change during play, by creating new blocks and by merging or capturing old ones. We assume that no new essential blocks are created during play, that merged blocks assume the identity of all constituent blocks, and that the liberty count function returns 0 for a captured block. In general, capturing an essential block finishes a semeai and all LCG associated with that block.

4.3. Pruning dominated moves in a LCG

It is possible to define a partial order of LCG by recursively testing whether the liberty counts in one game always dominate the other. Domination means that own blocks have at least the same number of liberties, while opponent blocks have the same or less. Given such a partial order, moves that lead to a worse LCG are dominated and can be pruned. Examples would be filling own liberties or eyes, or failing to extend liberties where that is possible.

5. Capturing races in Go

A *semeai* in the game of Go can be defined informally as "a race to capture between two adjacent groups that cannot both live". Fig. 5 shows two simple cases. Earlier work [7,8] contained more formal definitions of semeai, and described nine different classes. Semeai of classes 0, 1 and 2 can be detected and evaluated statically, without search. The other classes cover semeai that can be

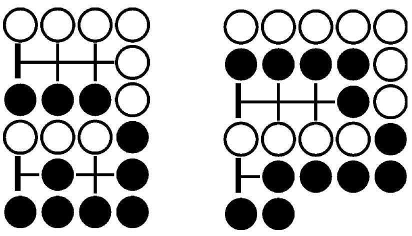


Fig. 5. Two simple semeai.

resolved by search, potential semeai, and unclear situations which might end up as a race to capture.

We only seek to determine the win/loss/seki outcome of a semeai. We do not consider other issues here, such as maximizing the score, computing the combinatorial game value, or determining whether winning a semeai is beneficial in the overall context of a game [7].

5.1. Describing semeai instances

Perhaps surprisingly, identification of semeai in a given Go position requires the same preliminary analysis as for the endgame [6]: identification of safe blocks and territories, followed by a partition of the rest of the board into connected components called *local games*. Each local game, consisting of empty points, plus possibly unsafe stones of either player, can potentially become a semeai. This holds even if the area is currently completely empty or contains stones of only one player.

5.2. Classification of blocks and empty points

Classification of points in a local game is a first step in identifying semeai. In each local game, we recognize the following types of blocks of stones and empty points:

Essential block: A block of black or white stones which must be saved from capture. Capturing an essential block immediately decides a semeai.

Nonessential block: Block which can be captured without losing the semeai. An example of a nonessential block is a small block contained in the opponent's eye.

Unknown block: Contains all remaining blocks that cannot be classified as either essential or nonessential blocks. Saving or capturing such blocks has some priority as a heuristic, but it does not necessarily decide the semeai.

Outside liberty: An empty point that is a liberty of an essential block of one player, but not a liberty of an essential opponent block. An outside liberty is called *plain* if it is also adjacent to a safe opponent block, so the opponent can fill the liberty without making additional approach moves.

Shared liberty: Common liberty of essential blocks of both Black and White.

Eye: An area completely surrounded by essential blocks of one player. The area can contain nonessential blocks of either player. A *plain eye* has only one surrounding block, and all empty points inside are adjacent to that block. This definition is broader than the usual one, and includes cases where the surrounded region will end up as more or less than one eye.

Unknown area: Area that cannot be classified as outside liberties, shared liberties, or eye.

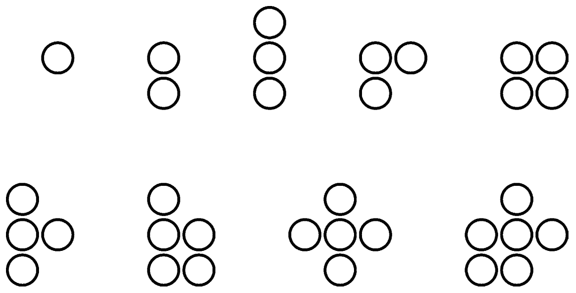


Fig. 6. Basic nakade shapes.

Table 1
Eye status and liberties

	Size							
	0	1	2	3	4	5	6	7
Status	0	1	1	1	4	5	6	7
Liberties	0	1	2	3	5	8	12	17

5.3. Eye status and liberty count

An eye area is called a *nakade* if the opponent can fill all but one of its points by one of the basic nakade shapes shown in Fig. 6.

In semeai, small eyes with size from 1 to 3 behave in the same way, while larger eyes are stronger both in terms of providing more liberties than their size and in having an advantage in semeai against smaller eyes. We model this behavior by an *eye status*. For each eye size, Table 1 shows the status and the number of liberties. For $0 \leqslant m < n \leqslant 7$, a n point nakade shape filled by m opponent stones is equivalent to $(n^2 - 3n)/2 + 3 - m$ outside liberties. A nakade shape is *unsettled* if it has not yet been reduced to only one eye, and the defender can still make two eyes there.

5.4. A general semeai algorithm

The following steps are a general outline of a semeai solving algorithm.

1. *Board partition:* Find safe blocks of stones and territories. Partition the rest of the board into connected components, called *local games*.
2. *Semeai identification:* Investigate which local games are semeai candidates by the following substeps.
   2.1. *Eye recognition:* Subdivide each local game into regions surrounded by stones of a *single* player. Test each such region whether it is a plain eye for that player.

   2.2. *Liberty regions:* After finding blocks and eye regions, divide the rest of a local game into liberty regions surrounded by stones of *both* players. Classify liberty regions as outside liberties, shared liberties, or unknown.
   2.3. *Block classification:* Classify blocks as essential blocks, nonessential blocks, and unknown blocks.
   2.4. *Semeai safety test:* For each color, determine if winning the semeai would ensure the safety of all essential blocks.
   2.5. *Semeai classification:* Determine which semeai class the local game belongs to.
3. *Static evaluation:* For semeai of classes 0 to 2, statically evaluate the semeai to find its status and its combinatorial game evaluation.
4. *Search:* For semeai of classes 3 or higher, use search to find the outcome.
5. *Move generation for semeai play:* Using the results of search or static evaluation, generate moves to play each semeai on the board. Use exact combinatorial game values when available, and a heuristic temperature estimate otherwise.

CCG can be used as an abstract representation of play in a local region that is part of a semeai. In some states nonlocal information is required to determine whether a move is possible. For example, the last liberty in an eye can be taken only as the last overall liberty of a block surrounding the eye.

5.5. Examples of LCG and contexts in Go

5.5.1. Plain outside and shared liberties

The game $G_n = (P_n, L)$ consisting of a single Black block b with n plain outside liberties can be defined by $P_0 = 0$, $P_{n+1} = \{|P_n\}$ and $L(b, P_n) = n$. Similarly, a plain shared liberty region between Black block b and White block w is defined by the LCG $G_n = (S_n, L)$ with $S_0 = 0$, $S_{n+1} = \{S_n|S_n\}$ and $L(b, S_n) = L(w, S_n) = n$.

5.5.2. Two-eyed group

A two-eyed single-block group g can be described by the LCG $E2 = (0, L)$ with $L(g, 0) = 2$.

5.5.3. Large eyes

The reason that large eyes are so valuable in semeai is their ability to provide extra liberties late in a fight, and force the opponent to fill shared liberties first. A characteristic of the different eyes is how many moves are left after the liberty count goes down to 1 for the first time. This is the crucial point since it contains the conditional move 0_{L0}. Fig. 7 shows such a sequence, starting from a seven point eye.

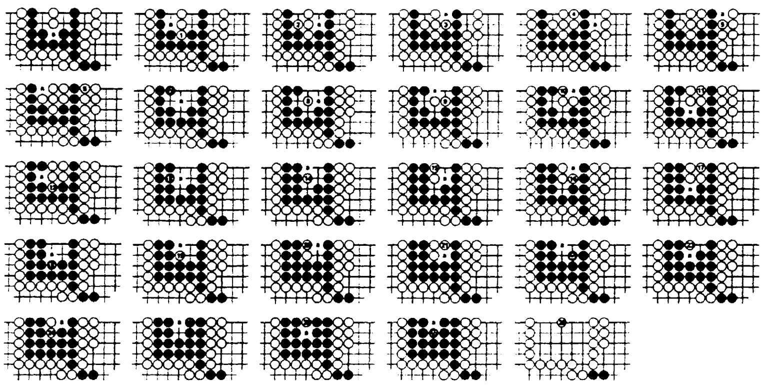
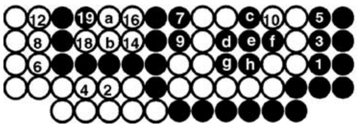


Fig. 7. Seven point nakade filling sequence.



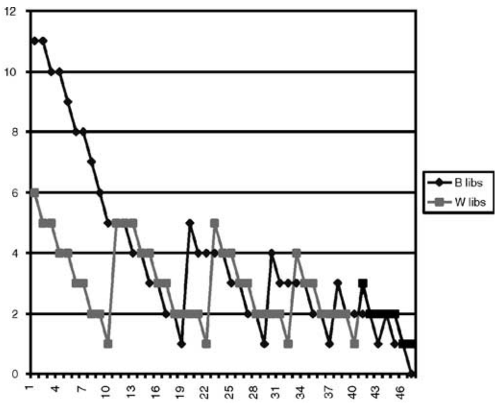


Fig. 8. Seven point eye vs six point eye. 11 at e, 13 at d, 15 at h, 17 at g, moves from 20 at b, c, f, e, 14, h, a, d, 16, 18, a, g, c, d, 16, e, b, 14, a, g, h, d, 16, b, a, e, 16.

Figs. 8 and 9 show the liberty counts during two long semeai sequences, each involving a seven point eye. In Fig. 8, initially Black has a six point eye containing two white stones and five outside liberties, while White has a seven point eye containing six black stones and three outside liberties. There are two shared liberties. The figure shows Black's failed attempt to capture White. Up to move 6, both remove outside liberties. With moves 7 and 9, Black fills the shared liberties since there are no other liberties that can be played. White is in atari and must capture with 10, reducing the area to a six point eye. After move 13, both have no outside liberties and a six point eye containing two opponent stones. In this balanced situation the first player can win by one move. In this case it is White. Both players' liberty count sequences are in lockstep from now on, and White remains one move ahead until capture.

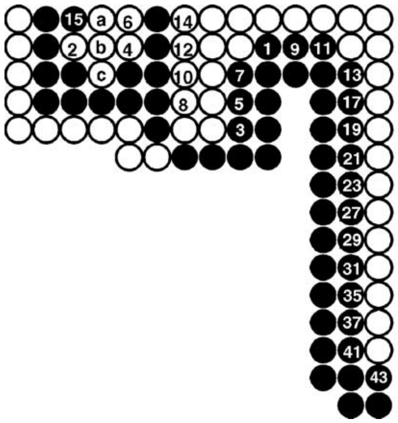
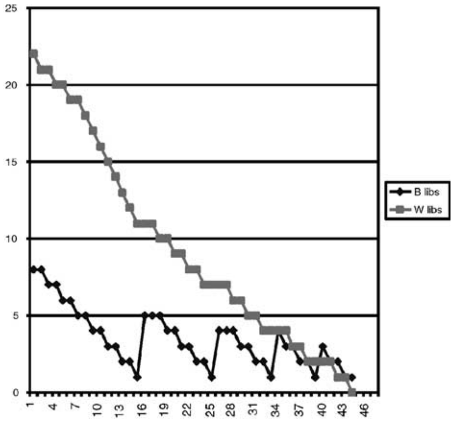



Fig. 9. Seven point eye vs no eye. 16 at b, 18 at 4, 20 at a, 22 at c, 24 at 2, 25 at 6, 26 at b, 28 at c, 30 at a, 32 at 2, 33 at 4, 34 at b, 36 at c, 38 at a, 39 at 2, 40 at b, 42 at c.

Fig. 9 pits a seven point eye against an eyeless group with many liberties. Up to 7, Black fills outside liberties and White fills Black's eye space. From 8 to 14, White fills the four shared liberties.

5.5.4. Protected liberties

Protected liberties have properties halfway between outside liberties and eyes. Protected liberties can be occupied directly only if $L0$ holds, but require one or more *approach moves* otherwise.

Fig. 10 shows a protected liberty of the block X. The CCG are $G4 = \{L3|0_{L0}, G3\}$, $G3 = \{L2|0_{L0}, G2\}$, $G2 = \{L1|0_{L0}, G1\}$, and $G1 = \{|0\}$. The liberty count function is $L(X, G) = 1$ for $G \in \{G1, \ldots, G4\}$, $L(X, Ln) = n$ and $L(X, 0) = 0$.

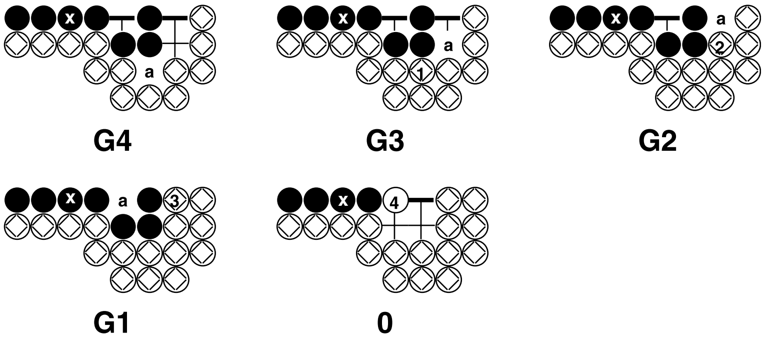


Fig. 10. Protected liberty.

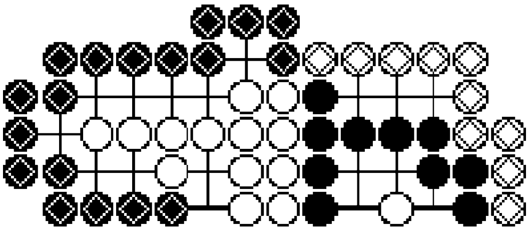


Fig. 11. Semeai test problem D, from [8].

5.6. Bounds

Instead of computing an exact LCG, it may be easier to determine the winner of a semeai by using bounds. As a trivial example, having any combination of *n* outside liberties is at least as good as having *n* plain liberties, but it may be better because the opponent may need extra approach moves and/or the player may have eye-making potential in the region.

Example: Fig. 11 shows problem D from Fig. 14 of [8], which was solved there by the search method of *partial order bounding*. We will show how to solve problem D statically by using bounds. Black has three outside liberties and an eye status of 5 with one opponent stone inside. Because there are no shared liberties, this is equivalent to a plain liberty filling sequence of 10 moves. White has eight plain liberties on the right and bottom, but White's eye space is unsettled on the left side. However, White has two nonplain liberties there, so White can win going first. In this case, creating an eye would be a fatal mistake for White. However, in other circumstances where Black does not have a large eye and where there are shared liberties, creating an eye would be the only good move.

In terms of partial order evaluation, we extend the evaluation of LCG by defining new games representing upper and lower bounds on real games as proposed in [8].

6. Related work

Ko fights in Go are an example where local play crucially depends on nonlocal context. The combinatorial game technique of thermography [2] has been extended to handle ko fights [1,9]. However, this approach still aims at abstracting away the specific context of a game and analyzes local positions in idealized abstract contexts such as *rich environments*.

Another case of nonlocal context described by Cazenave is the automatic generation of databases of Go patterns with external conditions [3]. These external conditions describe when a pattern for eye shape or life and death is valid and can be used by a program. Many Go programs use similar hand-generated patterns with external conditions, mainly concerning the number of liberties of blocks on the pattern boundary.

In his dissertation, Mäser [5] develops a model of local combinatorial games that contain global threats, which lead to an immediate win in any sum of local games. This could be modeled in a CCG framework by making all moves depend on a test for the execution of the global threat.

7. Summary and future work

We introduced the concepts of *CCG* and *LCG* as tools for the local analysis of semeai. We have shown how to integrate nonlocal aspects such as the total liberty count and ko status into such a framework as context conditions, and have given some examples to demonstrate that this work is a generalization of our previous work on semeai [7,8].

Future work includes working out the details of an implementation and of the local search process, and researching the overall strategy for selecting appropriate local analyses in complex semeai situations. CCG could be applied to other subproblems of Go, especially to endgame analysis, and to other games, such as Amazons.

References

[1] E. Berlekamp, The economist's view of combinatorial games, in: R. Nowakowski (Ed.), Games of No Chance: Combinatorial Games at MSRI, Cambridge University Press, 1996, pp. 365–405.
[2] E. Berlekamp, J. Conway, R. Guy, Winning Ways, Academic Press, London, 1982, Revised version published 2001–2002 by AK Peters.
[3] T. Cazenave, Generation of patterns with external conditions for the game of Go, in: H.J. van den Herik, B. Monien (Eds.), Advances in Computer Games 9, Maastricht and Paderborn, 2001, pp. 275–293.
[4] J. Conway, On Numbers and Games, Academic Press, London/New York, 1976.
[5] F. Mäser, Divide and Conquer in Game Tree Search: Algorithms, Software and Case Studies, PhD thesis, ETH Zurich, 2001.
[6] 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.
[7] M. Müller, Race to capture: Analyzing semeai in Go, in: Game Programming Workshop in Japan '99, IPSJ Symposium Series, vol. 99(14), 1999, pp. 61–68.
[8] M. Müller, Partial order bounding: A new approach to evaluation in game tree search, Artificial Intelligence 129 (1–2) (2001) 279–311.
[9] W. Spight, Extended thermography for multiple kos in Go, in: H.J. van den Herik, H. Iida (Eds.), Computers and Games. Proceedings CG'98, Lecture Notes in Computer Science, 1558, Springer Verlag, 1999, pp. 232–251.