Games of No Chance
MSRI Publications
Volume **29**, 1996

Experiments in Computer Go Endgames

MARTIN MÜLLER AND RALPH GASSER

Abstract. Recently, the mathematical theory of games has been applied to late-stage Go endgames [Berlekamp and Wolfe 1994; Wolfe 1991]. Based upon this theory, we developed a tool to solve local Go endgames. We verified all exact game values in [Wolfe 1991] and analyzed some more complex positions. We extended our method to calculate bounds for positions where optimal play depends on Ko.

Our program Explorer uses this tool to play full board endgames. It plays a nontrivial class of endgame positions perfectly. In a last section we discuss heuristic play for a wider range of endgames.

1. Go Endgames

Towards the end of a Go game, the board position usually breaks down into several local fights that can be analyzed individually (Figure 1). To find an optimal move globally, one needs to consider relations between these local subgames: There is a conflict between maximizing the local score and gaining the initiative to play in another part of the board. These relationships can be very complex, even surpassing the abilities of professional Go players. Exhaustive analysis of

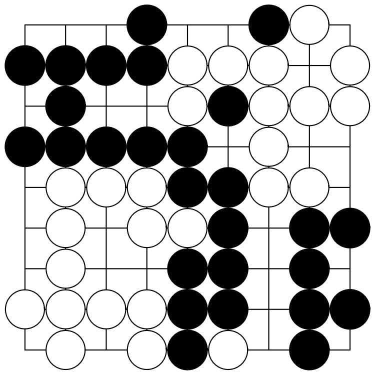


Figure 1. Late endgame position suitable for exact analysis.

full board endgames is only feasible for the most trivial problems, because of exponential explosion of the search.

The mathematical theory of games [Conway 1976; Berlekamp et al. 1982] can handle such complexity by divide and conquer: It defines an operation called a sum of games that builds up a game from its subgames (if no Ko fights exist, as explained below). This theory gives us algorithms to prune bad moves, compare the value of games and find an optimal move without exhaustive search on the full board.

To apply the theory to a Go endgame, we must assume that no Ko fights are left. (Extensions to handle some Ko situations will be discussed later.) By convention, positive scores are good for Black, negative scores are good for White. We use Japanese rules (though Chinese rules yield similar result). See [Berlekamp 1991; Berlekamp and Wolfe 1994] for a discussion of scoring rules.

The mathematical theory of games defines four basic types of games G:

$G > 0$: Black wins, no matter who plays first.
$G < 0$: White wins, no matter who plays first.
$G = 0$: Null game: the first player loses.
$G \parallel 0$: The first player wins.

Null games in Go are areas where no player can make a profitable move, like seki under Japanese rules. Games with an integer value—for example, safe territories—are also considered played out.

2. The Endgame Algorithm

We have implemented a six-step algorithm for endgame play:

- Board partition: find safe blocks, safe territories, and endgame areas.
- Generate local game trees in each endgame area.
- Evaluate local terminal positions (surrounded points plus prisoners).
- Transform local game trees into mathematical games, and simplify games.
- Calculate game as sum of local games.
- Find an optimal move in the sum game and play it.

Determining safe blocks, safe territories and endgame areas. We use an extension of the algorithm in [Benson 1980] to determine safe blocks and territories. Territory may contain prisoners if they have no chance of living. The difference in size of territories yields the constant part of the game value. See Figure 2, where Black has six points of safe territory and White has four, so $G_{\text{const}} = 6 - 4 = 2$.

Next we compute connected components of the remaining unsafe blocks and empty points. Each component constitutes an endgame area. Play in one area cannot affect play or the score in other areas, because the areas are separated by safe blocks (Figure 3).

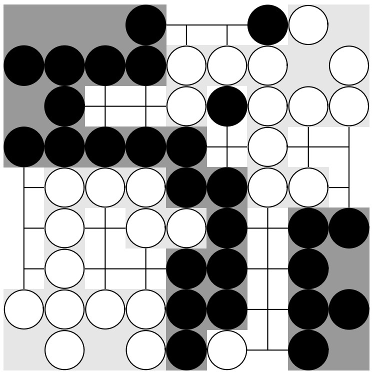


Figure 2. Safe blocks and territories.

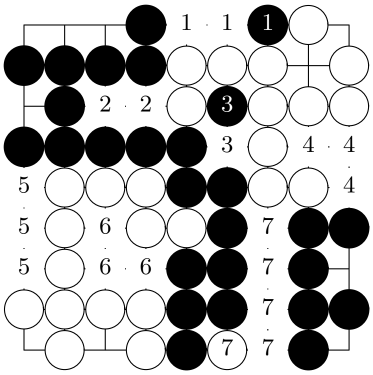


Figure 3. Endgame areas.

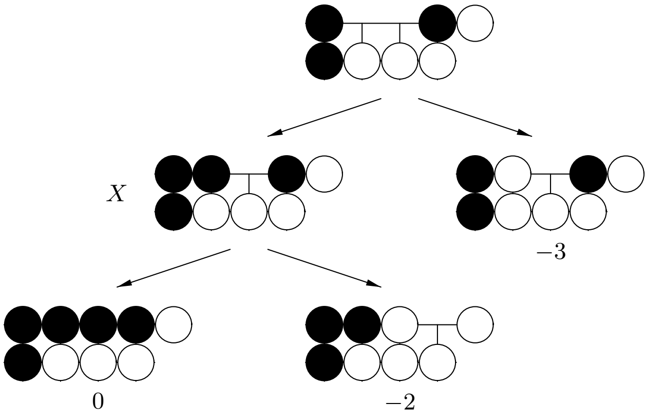


Figure 4. Local game tree for G_1.

Generating local game trees and evaluating terminal positions. In contrast to global game trees, local game trees must contain successive moves by the same color, because the opponent might tenuki (play in another local game). Move generation is locally exhaustive, with some rules for pruning dominated and reversible moves. Terminal positions are positions without a good move: all points are occupied or have become territory. We compute the local score for each terminal position (Figure 4).

Transforming game trees into mathematical games. Each local game tree is then transformed into a mathematical game, as follows. First sort all moves from a certain position by color to get the left and right options. For each move to another position, insert the value of that position. We have the numeric value of all terminal positions and compute the other games' values from them. In the example above we obtain $G_1 = \{X \,|\, {-3}\}$ in a first step. In the second step we evaluate X to get $X = \{0 \,|\, {-2}\}$, and therefore $G_1 = \{\{0 \,|\, {-2}\} \,|\, {-3}\}$. The complete values in our example are:

$$G_1 = \{0\,|\,\{-2\,|\,-3\}\}, \quad G_4 = \{\{0\,|\,-1\}\,|\,-2\}, \quad G_6 = \{\{0\,|\,-1\}\,|\,-2\},$$
$$G_2 = \{1\,|\,0\}, \quad G_5 = \{\{0\,|\,-1\}\,|\,-2\}, \quad G_7 = \{5\,|\,\{4\,|\,\{3\,|\,\{2\,|\,0\}\}\}\}.$$
$$G_3 = \{0\,|\,-2\},$$

Calculating the sum of local games. The whole game is the sum of territory values plus all local games; in our running example, $G = G_{\text{const}} + G_1 + \cdots + G_7$. Even a sum of such short games can get very complicated: Printed out, G has several hundred characters. The operations can be simplified by chilling all endgames before computing the sum [Berlekamp and Wolfe 1994]. Chilling is a one-to-one mapping of endgames to simpler ones.

Finding an optimal move. Given a sum game and a player, it is easy to compute her minimax value. To find an optimal move, we try out all moves and select one that preserves the minimax value.

An alternative, usually more efficient, method uses the incentives of moves to find an optimal move.

3. Software

We implemented our endgame player on top of two existing programs: Wolfe's toolkit for mathematical games [Wolfe 1996] and our Go program Explorer [Müller 1995]. Wolfe's toolkit is written in ANSI C, Explorer in Modula-2. An interface between both programs was implemented as a term project [Fierz 1992]. The resulting program allows direct calls from Explorer to Wolfe's toolkit, as well as storage and retrieval of mathematical games in Explorer. For faster development we chose a low level of integration: Modula-2 and C parts keep their separate management of resources like memory blocks, lists and hash tables. The command line interface to the C toolkit was replaced by menu commands as needed.

4. Local Move Generation

Ignoring captures and illegal moves, the number of possible plays in an n-point area is approximately $2n$ (n for each player), and each play generates an $(n-1)$-area. A rough estimate for the size of the game tree is therefore $2^n n!$.

Due to combinatorial explosion, even fairly small endgames become prohibitively expensive to compute using this approach. We now turn to some techniques for reducing the size of the tree.

Hashing. There are at most 3^n different configurations on n points; this grows much more slowly than $2^n n!$. Thus, even a simple hash function will detect many transpositions in the game tree.

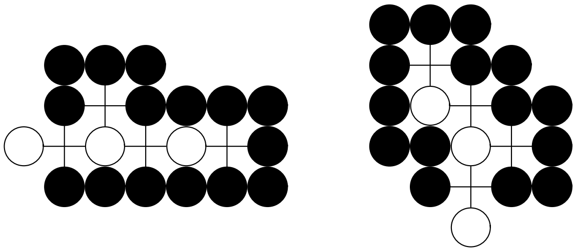


Figure 5. Isomorphic positions.

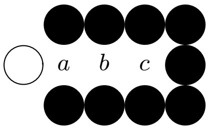


Figure 6. Here $a > b > c$.

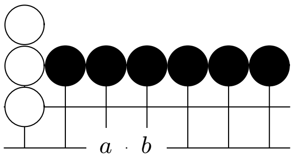


Figure 7. Only a and b appear useful.

An improved hash function might also recognize Go-specific graph isomorphisms, such as the one in Figure 5.

Pruning. The search tree can be further reduced by pruning. We can do this in either a rigorous or heuristic fashion. If we prune rigorously, we only eliminate moves that are provably worse than others (prune b if there is a such that $a \geq b$). For example, for corridors it can be shown that moves are better the closer they are to the entrance of the corridor. In Figure 6, we need therefore only consider move a for both players.

If we do not need the exact value of a position, heuristic pruning can be used to eliminate even more moves. For instance, in Figure 7, we may decide only to consider moves a or b for White.

Early Termination. If a position's value can be determined statically, the entire subtree can be pruned. For instance, any position that contains only dame points or safe territory can be easily evaluated in this fashion; Figure 8 shows an example. The value of n dame is 0 if n is even, and $\{0|0\} = *$ if n is odd.

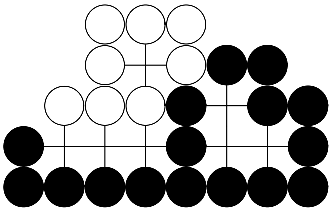


Figure 8. The value of this game is $2*$.

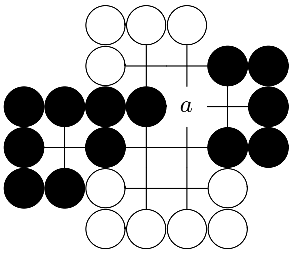
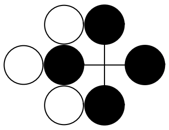


Figure 9. Black's play at a separates the area into three independent areas.

Splitting the local game. Often playing a move results in the local endgame splitting into several independent areas. In this case, we can analyze each area separately. The position's value is then the sum of the values of the independent areas. In Figure 9, if Black plays a the area separates into three independent positions, two of which are solved immediately, without further search (the black territory and the two dame points). Even if no areas could be pruned, this method still shrinks the game tree dramatically.

5. Bounds for Ko

Classical combinatorial game theory cannot handle loops in games. Unfortunately, loops often appear in Go—for example, in the case of Ko. We must therefore find a way of eliminating these loops from the game tree.



One way is to define the symbol K, which stands for the game at right. Whenever such a situation occurs in the game tree, we prune the loop and substitute the value K instead. This only masks the problem, since we still do not know how to play optimally. Also the simple Ko is not the only situation in Go that leads to loops. Assigning additional constants for all such situations does not seem feasible.

Another way of resolving this problem is to calculate bounds on the game value. This can be done by not considering all moves for a certain player. For instance, by suppressing any move by Black that would lead to a loop in the game tree, we calculate a lower bound. By suppressing White's options, we get an upper bound. If these two bounds are identical, optimal play does not depend on Ko and we have determined the exact game value. If the bounds differ, we may be able to determine which bound is more appropriate by considering the number of Ko threats still on the board.

In Figure 10, we determine the value of position A. The game tree has already been pruned, so that only the critical paths remain.

The following dependencies hold for the unknown game values:

$$A = \{2{*}\,|\,B\}, \quad B = \{C{+}2\,|\,0\}, \quad C = \{1\,|\,D{-}1\},$$
$$D = \{E\,|\,{*}, B\}, \quad E = \{F{+}1\,|\,0\}, \quad F = \{0\,|\,E{-}1\},$$

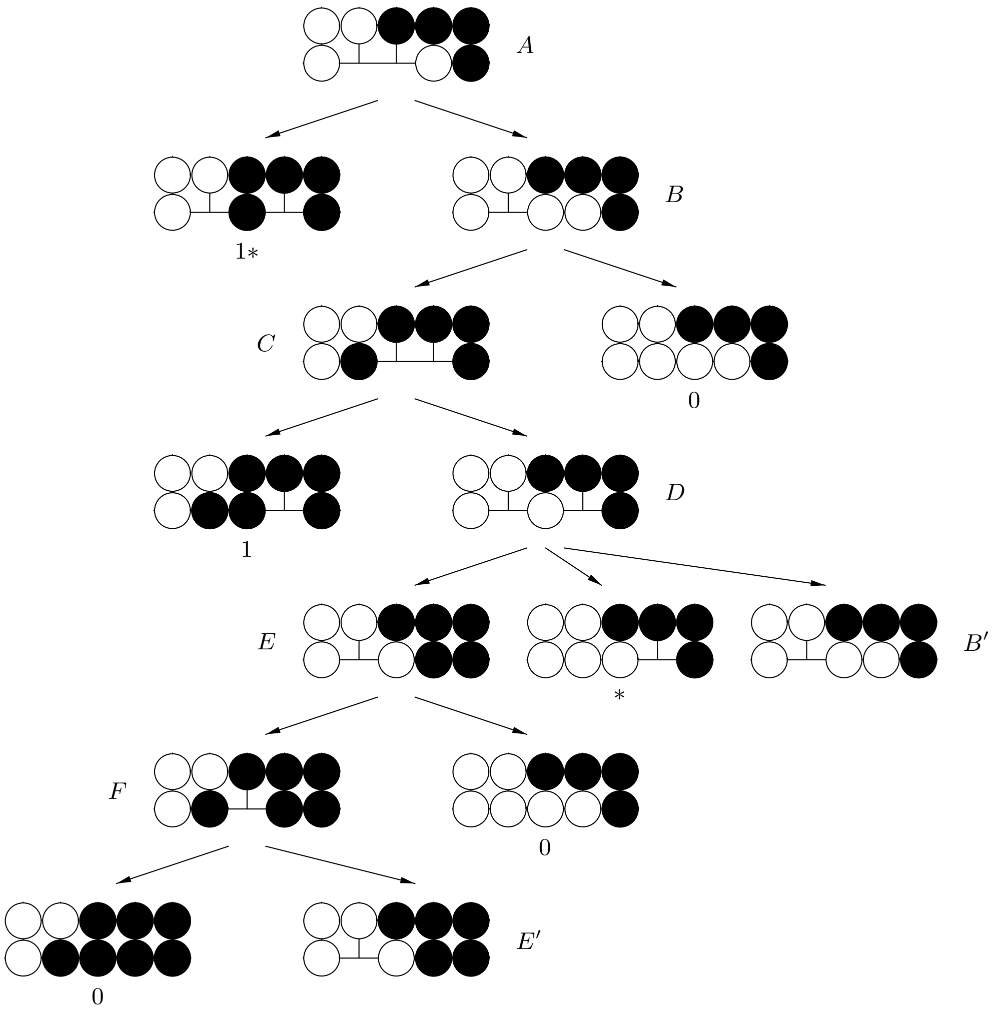


Figure 10. In this example, the calculation of bounds proves that Ko's do not affect the value of A.

We now determine a bound on the game value if White wins all Ko's. This is modeled by not allowing Black any moves that would lead into a repetition. So in E', Black is not allowed the move to F'; therefore

$$E' = \{\,|0\} = -1 \qquad F = \{0|-2\} \qquad E = \{1|-1\|0\} = -1.$$

Similarly, in B', Black's move to C' is forbidden, resulting in the following values:

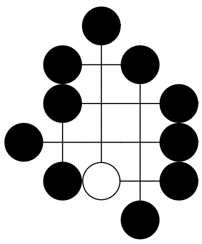
$$B' = \{\,|0\} = -1 \qquad D = \{-1|*,-1\} = -1* \qquad C = \{1|-2*\}$$
$$B = \{3|*\|0\} = * \qquad A = \{2*|*\}$$

As we will see, this is not necessarily the final answer for A. We must make another pass through the tree, but this time instead of cutting off a move for Black completely, we use the value for that position obtained in the previous pass:

$$E' = \{\text{prior } F{+}1|0\} = \{1|-1\|0\} = -1 \qquad F = 0|-2 \qquad E = \{1|-1\|0\} = -1$$
$$B' = \{\text{prior } C{+}2|0\} = * \qquad D = \{-1|*,*\} = \{1|*\} = 0$$
$$C = 1|-1 \qquad B = 3|1\|0 \qquad A = 2*\|\|3|1\|0 = 2*|1$$

The algorithm continues in this manner until no position's value changes compared to the last pass. For the above position the value is $2*|1$ regardless of whether White or Black wins Ko's. Thus, optimal play does not depend upon Ko.

In practice, this algorithm has always terminated. We are not sure however, if this is always the case. Possibly there are cases where this method does not converge, or does not converge to a fixpoint in finite time.



We have verified the values of all 103 *node room* positions in [Wolfe 1991]. For positions where optimal play does not depend on Ko, we obtain identical results. For the positions where optimal play depends on Ko, our program returns bounds. The most complex Ko position, depicted on the right, has upper bound $5|\{4\|\|3\|2|1*\}\|\|2|1*\|*$ (Black wins Ko's), and lower bound $\{5\|\|4\|3|*\}\|\|2|1*\|*$ (White wins Ko's). The cooled values are $\frac{49}{16}|\frac{7}{4}$ and $3|3{+_1}\|\frac{7}{4}$, respectively. The means, respectively $\frac{77}{32}$ and $\frac{76}{32}$, are only $\frac{1}{32}$ apart, and likewise the temperatures $\frac{53}{32}$ and $\frac{52}{32}$.

6. Heuristic extensions to endgame algorithm

Many real endgame positions are not suitable for exact analysis: The endgame areas may be too big, or there may be Ko's on the board. In these cases we need to use heuristics.

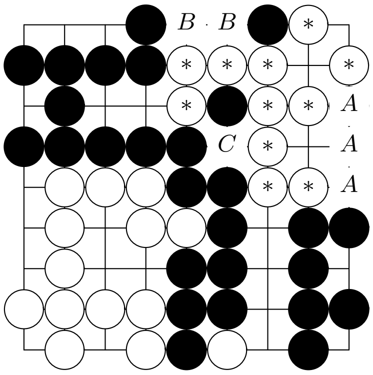


Figure 11. The stones marked $*$ die if Black gets all moves in endgames A, B and C.

Relaxed definitions of safety. The rules for determining safe stones and territories are very restrictive. This leads to many unsafe blocks and big endgame areas.

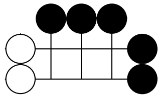
Figure 11 shows one of Berlekamp's endgame problems. It is the same as our running example, except for the absence of a stone at the uppermost A point. The stones marked $*$ are not completely safe: they will die if Black gets all moves in endgames A, B and C. Therefore there may be a weak relation between these endgames. In heuristic endgame play we can choose to ignore this relation, regard the marked stones as safe and independently analyze the areas A, B and C.

Heuristic move generation and local evaluation. In areas too big to analyze exhaustively, we need heuristic move generators which build a forward pruned subtree of an endgame. We also use a heuristic evaluation of nonterminal positions. By restricting the moves of only one player, we can still calculate correct upper and lower bounds for such positions, but they will be less tight than before.

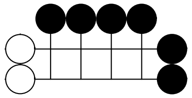
Playing for a win. Playing for a win (a fixed number of points) is often much easier than optimal play. A program may not be able to solve an endgame position completely, but it may come up with bounds that are good enough to prove a win by, say, half a point. Human experts often do this kind of analysis in tournament games with limited time.

7. Mathematical Value, Mean and Temperature of Some Common Endgames

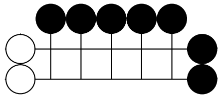
The diagrams on the next page show values or bounds for several corridors of width two and three. Expressions marked $\bullet$ and $\circ$ indicate bounds obtained under the assumptions that Black and White, respectively, wins Ko's.



$3|*$ cooled: $2|1$ $\mu = \frac{3}{2}$ $t = \frac{3}{2}$



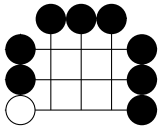
$5\|1*|0$ cooled: $4|\frac{3}{2}$ $\mu = \frac{11}{4}$ $t = \frac{9}{4}$



● $7|\{3*|0\}, \{7*|5*\|2*\|\|1*\}$ cooled: $6|\{3|2\}, 3_{-2*}$ $\mu = \frac{17}{4}$ $t = \frac{11}{4}$

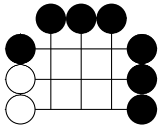
○ $7|\{3*|0\}, \{\{8\|\|7\|2, \{5|1\}|0\}|\{6*\|4|2\|\|1|-1\}, \{5, \{6, \{7|5\}|4\}|2*\|1*\}\|0\}$

cooled: $6|\{3|2\}, \{6+_{4|4,\{4|2\}}|\{5|4*\|2*\}, \{4, 4\uparrow*|3\|3\}\|2\}$ $\mu = \frac{17}{4}$ $t = \frac{11}{4}$



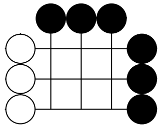
● $8|\{2*\|\|1*\|*|-1\}, \{7|\{4\|\|3\|2|0\}\|3|1\|\|*\}$ cooled: $7|\frac{17}{8}, \{5|3\uparrow2*\|3*\|\|2\}$ $\mu = \frac{73}{16}$ $t = \frac{55}{16}$

○ $8|\{5|2*\|1*\|\|*\}$ cooled: $7\|2_{-1}|2$ $\mu = \frac{9}{2}$ $t = \frac{7}{2}$



● $\{7\|\|5|\{4\|\|3\|2|0\}\|\{5|2*\|1*\|\|0|-2\}\}|0$ cooled: $\{5\|\|3\uparrow3\|2_{-1}|1*\}|1$ $\mu = \frac{9}{4}$ $t = \frac{9}{4}$

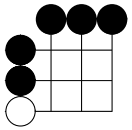
○ $5|\{4\|\|3\|2|0\}\|0$ cooled: $3\uparrow3|1$ $\mu = 2$ $t = 2$



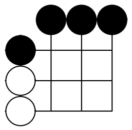
● $\{4|3*\|\|3|2*\|1|*\}, \{\{6\|\|5\|1*|0\}|\{1*|0\}, \{4*|3\|*|-1\}\}\|*|-1$

cooled: $\frac{3}{2}\uparrow, \{3+_{\frac{5}{2}}|\frac{1}{2}, \{\frac{5}{2}|\frac{1}{2}\}\}|\frac{1}{2}$ $\mu = 1$ $t = \frac{3}{2}$

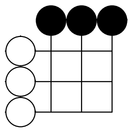
○ $\{3\|\|2\|1|*\}\|*|-1$ cooled: $\frac{9}{8}|\frac{1}{2}$ $\mu = \frac{13}{16}$ $t = \frac{21}{16}$



• $\{7\|3*|2\}, \{7|\{6\||1\|0|-3*\}\}$
$|\{\{6\||1\|0|-3*\}\||1, \{6, \{7|1\}|0\}|-1\|-3*\},$
$\{\{7\||4\|2*|1\}, \{7\|4|\{3\||1|-5\|-6\}\}$
$|\{2*|1\}, \{3\|1*|0\}, \{3, \{4\|2*|1\}$
$|\{2*|1\|\{1*|0\||*|-3\|-4\}, \{*, \{3|2*\|\{3|1\|-2\||-3\}\}\||*|-3\|-4\}\}, \{3\|1*|0\||*|-5\}\}\|-3\}$
cooled: $\{5|\tfrac{5}{2}\}, 5{+_{4-1}}|\{5|1{+_1}\||1, \{5, \{5|1\}|1\}|1\|0\},$
$\{\{5\|3|\tfrac{5}{2}\}, \{5|3{+_{6+4}}\}|\tfrac{5}{2}, \{3|\tfrac{5}{2}\}, \{3, \{3|\tfrac{5}{2}\}|\{\tfrac{5}{2}|\{\tfrac{5}{2}|1{-_1}\}, \{2, \{\tfrac{7}{2}|1{-_{2*}}\}|1{-_1}\}\}, \{3|\tfrac{5}{2}\|2|-1\}\}\|-1\}$
$\mu = 3 \quad t = 3$
∘ $7\|3*|1*\||\{5|1\|0\||*|-1\|-3*\}$ cooled: $5|2*\||1{-_2}\|\tfrac{1}{2}|0 \quad \mu = 2 \quad t = \tfrac{19}{8}$



• $7|\{5\|3*|2\||\{5|1\|0\||\{*|-2*\}, \{6|0\|-3\}|-4, \{-3|-5\}\}\}$
$\|\{*|-1\}, \{5|1\|0\||\{*|-2*\}, \{6|0\|-3\}|-4, \{-3|-5\}\},$
$\{3*, \{4*|2*\}|\{2*|\{1*\||0|-4\|-5\}, \{0, \{5|3\|2\||-3\}\||0|-4\|-5\}\},$
$\{3*|1*\|\{1\|0|-4\||-6\}\}\|\{5*|\{5|3\|2\||-3\}\|-3, \{1\|0|-4\||-6\}\||-4\}\}$
cooled: $5|\{3|\tfrac{5}{2}\||1{-_2}\|*, \{5|1\|-1\}|-1, -1*\}$
$\|\tfrac{1}{2}, \{1{-_2}\|*, \{5|1\|-1\}|-1, -1*\},$
$\{2, 2*|\{2|\{2|-1{-_2}\}, \{1, \{3{\downarrow}|-1\}|-1{-_2}\}\}, \{2*\|1{+_2}|-3\}\|\{4\|3{\downarrow}|-1\||-1, \{1{+_2}|-3\}\}|-1\}$
$\mu = \tfrac{3}{2} \quad t = \tfrac{9}{4}$
∘ $5\|3*|1*\||\{0|-1*\|-3\}, \{3*|\{1|-1\}, \{2*|-4*\}, \{\{2\||1\|0|-6\}\|-1*|-4\}\||-1|-3\|-4\}$
cooled: $3|2*\|\{-\tfrac{1}{2}|-1\}, \{2|1*, \{2|-2\}, \{1|1{+_4}\|0|-1\}\|-1{\downarrow}\} \quad \mu = \tfrac{7}{8} \quad t = \tfrac{21}{8}$



• $\{3|\{2, \{3|1\}\||2|0\|-7\}\}, \{6\|5|*\||*, \{4*|2*\||0|-1*\|-3\}\}\||0|-1*\|-3$
cooled: $\{1\||1, 1*\|1*|-5\}, \{3{+_3}|0, \{2*\|-\tfrac{1}{2}|-1\}\}\|-\tfrac{1}{2}|-1 \quad \mu = \tfrac{1}{8} \quad t = \tfrac{15}{8}$
∘ $3\|1*|0\||\{\{7\|0|-2\||-2, \{-1|-3\}\}|-3\}, \{*, \{3\|1*|0\||-2*|-4*\}\||*|-5\|-6\}$
cooled: $1|\tfrac{1}{2}\|\{5|-1*\|-1, -1*\||-1\}, \{0, \{1|\tfrac{1}{2}\|-2*\}|-3{-_3}\} \quad \mu = -\tfrac{1}{8} \quad t = \tfrac{15}{8}$

References

[Benson 1980] D. B. Benson, "A mathematical analysis of Go", pp. 55–64 in *Proc. 2nd Seminar on Scientific Go-Theory* (edited by K. Heine), Institut für Strahlenchemie, Mühlheim a. d. Ruhr, 1979.

[Berlekamp 1991] E. Berlekamp, "Introductory overview of mathematical Go endgames", pp. 73-100 in *Combinatorial Games* (edited by R. K. Guy), Proc. Symp. Appl. Math. **43**, Amer. Math. Soc, 1991.

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

[Berlekamp et al. 1982] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways For Your Mathematical Plays, I: Games In General*, Academic Press, London, 1982.

[Conway 1976] J. H. Conway, *On Numbers And Games*, Academic Press, London, 1976.

[Fierz 1992] W. Fierz, "Go Endgames", Semesterarbeit, ETH Zürich, 1992.

[Müller 1995] M. Müller, "Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory", Ph.D. Dissertation, ETH Zürich, 1995.

[Wolfe 1991] D. Wolfe, "Mathematics of Go: chilling corridors", Ph.D. Dissertation, Univ. of California, Berkeley, 1991.

[Wolfe 1996] D. Wolfe, "The Gamesman's Toolkit", pp. 93–98 in this volume.

Martin Müller
Institut für Theoretische Informatik
Eidgenössische Technische Hochschule (ETH)
8092 Zürich
Switzerland

Ralph Gasser
Institut für Theoretische Informatik
Eidgenössische Technische Hochschule (ETH)
8092 Zürich
Switzerland