Counting the Score: Position Evaluation in Computer Go

Martin Müller
Department of Computing Science, University of Alberta
Edmonton, Canada T6G 2E8
mmueller@cs.ualberta.ca

Abstract

Position evaluation is a critical component of Go programs. This paper describes both the exact and the heuristic methods for position evaluation that are used in the Go program *Explorer*, and outlines some requirements for developing better Go evaluation functions in the future.

1 Introduction

The standard approach to developing game-playing programs is minimax search using a fast full-board evaluation. However, in many early Go programs, position evaluation played only a minor role. Programs selected their moves without ever computing a full-board evaluation. Moves were generated by very selective, goal-oriented move generators which used heuristics to approximate the value of a move. Lack of speed and quality problems were the two main reasons for deviating from the standard approach in the game of Go: full-board position evaluation is complicated and slow, so early programs did not have the computer resources needed to perform any lookahead. The quality of play also played a major role: the effect of many good moves in Go is hard to measure by a position evaluation function. Examples are creating "good shape" and many kinds of necessary defensive moves. Unless the evaluation function is incredibly sophisticated, and can reliably handle subtle changes in the strength of stones, it will miss the effect of such moves. Typically, they do not increase the territorial score of a Go position by much, or might even seem to decrease it in case a move defends against a hidden threat. In the author's experience [9], a purely territorial evaluation leads to "greedy" play and overextensions. As described by Chen [7], many programs use a mix of move evaluation and position evaluation. For example, a program based on position evaluation can give a bonus or a penalty to those types of moves that are otherwise misevaluated.

Several surveys on computer Go [5, 6, 15, 20] have described the many components of current programs, with an emphasis on knowledge representation and search techniques. Papers dealing with specific position evaluation methods are [3, 7, 19, 21, 22]. This article describes the position evaluation methods used in the author's program

Explorer. There are two main differences to previously described approaches: a modularized heuristic evaluation using the concept of *zones*, and an emphasis on exact evaluation methods for several types of local situations.

The structure of this paper is as follows: Section 2 describes three exact evaluation methods for safe territories, capturing races and endgames that are used in Explorer. Section 3 deals with Explorer's zone-based heuristic position evaluation. Section 4 shows how a full-board position evaluation is computed from the different local evaluations, and Section 5 discusses open problems and future work on position evaluation.

2 Exact Evaluation in Explorer

This section describes three types of local situations for which Explorer is able to compute an exact evaluation: safe territories, endgames, and capturing races (*semeai*).

2.1 Safe Territory

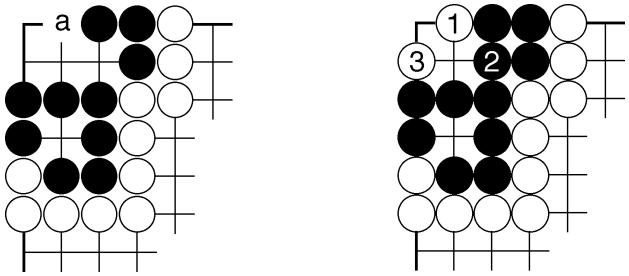

Figure 1: Safe stones, unsafe territory.

Safe territories are parts of a Go position that completely belong to one player. They cannot be reduced or destroyed by the opponent under normal circumstances. Recognizing safe territories is useful for proving that a game is over. It is also required for exact mathematical endgame analysis, which starts by partitioning the board into small independent endgame areas divided by safe blocks [9, 11]. Safe territories are games with a constant integer value.

Determining the safety of a completely surrounded territory, and the stones surrounding it, is similar to solving Life and Death problems. One main difference is that safety proofs for large areas cannot use an exhaustive search since the state space is too large. Another difference is in the treatment of coexistence in *seki*. While stones are safe if the opponent cannot capture them, territory is safe only if it can be proven that no opponent stones can survive on the inside. Figure 1, from [10], shows an example where the black stones are safe but the area that they surround is not. White 1 threatens to capture three black stones, so Black 2 is forced. After White 3, the final result is coexistence in a seki, and the black territory has been destroyed.

Benson [1] has given a mathematical characterization of *unconditionally alive* blocks of stones. Such blocks can never be captured, not even by an arbitrary number of successive opponent moves. For example, all black blocks in Figure 2 are unconditionally

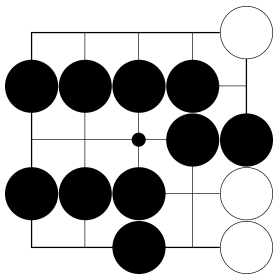

Figure 2: Benson's unconditional safety.

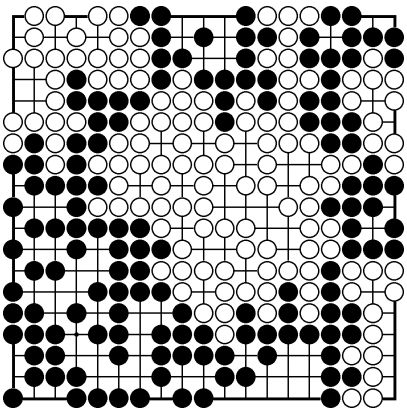

Figure 3: Proving the safety of blocks and territories.

safe. All the black stones always have at least two adjacent empty squares, and White cannot fill them without committing suicide. [10] introduced both static rules and methods based on local search for detecting groups of blocks that are safe under the usual alternating play, where the defender is allowed to reply to the attacker's moves. These techniques can be used to prove the safety of many moderately large areas. In well-subdivided positions near the end of a game, such as the example shown in Figure 3, every point on the board can quickly be proven safe with current techniques. Explorer implements both Benson's algorithm and an enhanced version of the techniques described in [10].

2.2 Semeai - Capturing Races

Capturing races (semeai) are a type of local Go situations where very specific, strong rules for evaluation and efficient search exist [12, 14]. Explorer can exactly evaluate two kinds of capturing races:

1. Semeai that have already been decided in one player's favor. The value is an

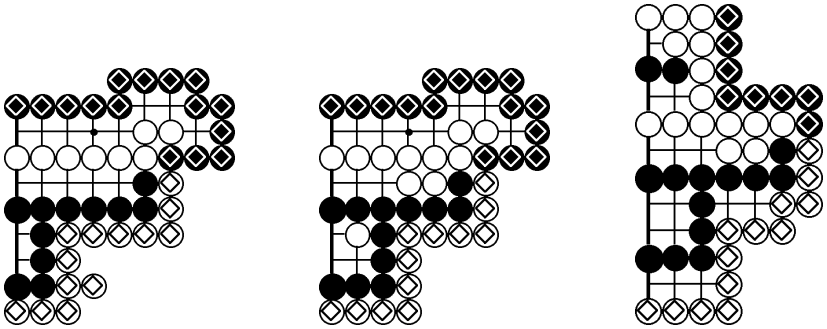

Figure 4: Semeai won by one player, evaluated statically by Explorer.

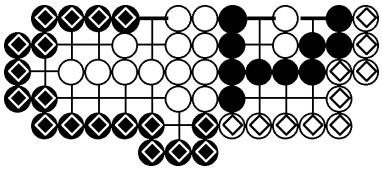

Figure 5: Semeai won by one player, proven through search by Explorer.

integer, the size of the whole area, with the possible exception of a number of neutral points on the outside. These semeai are equivalent to safe territories, but cannot be recognized by the usual methods for finding safe territories. Explorer can find such semeai by static rules, as in the examples of Figure 4, or by a search, as in the example given in Figure 5.

2. For semeai of simple structure, namely the classes 0, 1 and 2 defined in [12], Explorer can compute the exact combinatorial game value. Many of these semeai values are simple switches, while others involve up to four different relevant outcomes. See Figure 6 for some examples.

2.3 Endgame Areas

Combinatorial game theory provides a range of endgame analysis techniques that compute the "best possible" evaluation at different levels of precision. Many of these techniques are implemented in Explorer. For endgames without ko fights, the so-called canonical form of a game can be computed by a local search [11]. This method yields complete information about good local plays. A more compact representation of local game values are thermographs. In a sense that can be made mathematically precise, the *mast value* of a thermograph is the best possible static evaluation of an endgame area by a single number [2]. Similarly, the *temperature* of a thermograph is a measure of urgency of playing a move in that area.

The theory of thermography has been extended by Berlekamp and Spight to handle

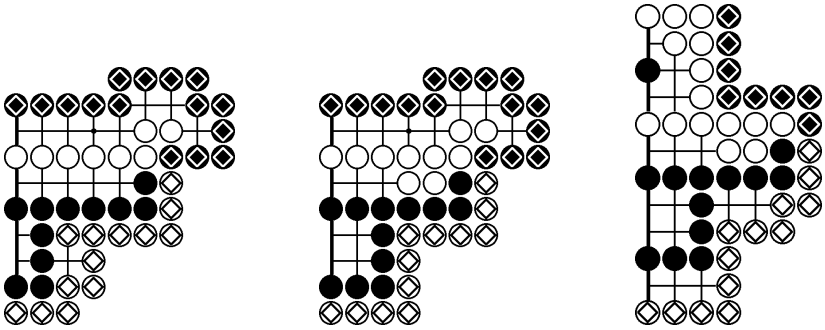

Figure 6: Semeai test problems A, B and C, from [14].

Ko fights [2, 17, 18]. Explorer can compute the thermograph values of local positions involving a single ko at a time [13]. Bill Fraser's programs *GoSolver* and *BruteForce* can analyze more complex ko situations [8].

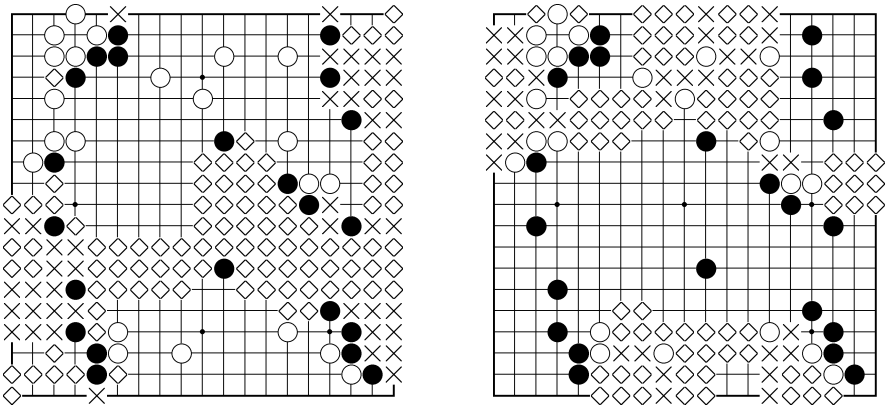

Figure 7: Dividers ($\times$) and potential dividers ($\diamond$) for Black and White.

3 Heuristic Territory Evaluation in Explorer

This section describes the territory-related features of the program Explorer [9]. Their combination into a full-board score is discussed in Section 4.

For evaluation purposes, Explorer distinguishes several different types of points. The intermediate data structures and final results of the computation are shown for a sample position from a recent computer-computer game in Figures 7 to 11. *Zones*

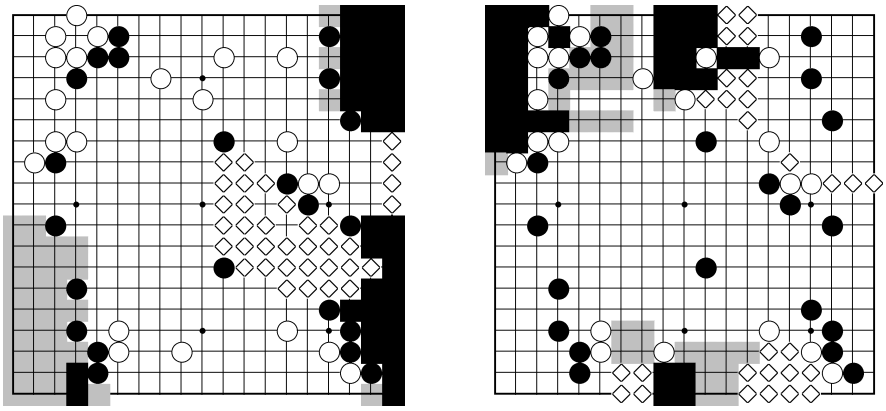

Figure 8: Safe (dark shade), potential (light shade), and threatened (⋄) zones.

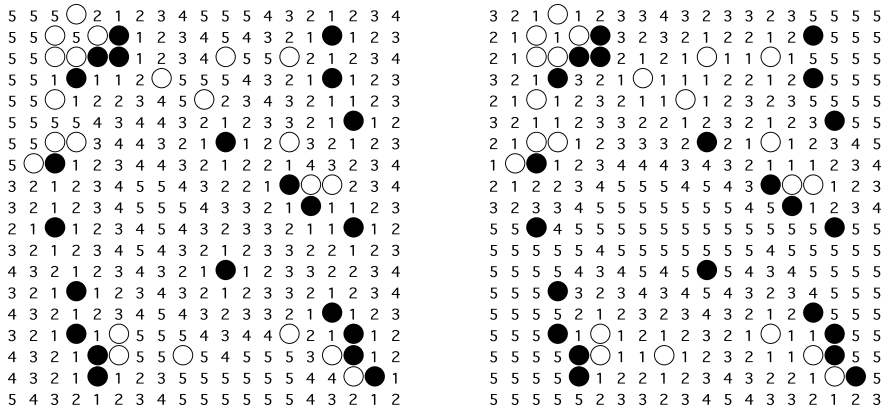

Figure 9: Divider-adjusted distances from Black and White stones, with maximum distance 5.

(Figure 8) are computed from blocks of stones, *dividers* and *potential dividers* (Figure 7). Using heuristic information such as the map of connectible points shown in Figure 11, zones are classified as *safe territory*, *potential territory*, *threatened zone*, or *unused zones*. Points outside of territories are further classified into *near points*, *junction points*, and *far-away points*, as shown in Figure 10. Details of the process are described below.

In Explorer, position evaluation is performed in the later stages of the overall position analysis. It requires most of the previously computed information as an input. Such information includes the results of tactical analysis, life and death searches, and complex heuristic rules to identify alive, weak and dead groups of stones [9]. Position evaluation itself is a four step process:

1. *Dividers* and *potential dividers* are computed using pattern matching. Each divider or potential divider is a small object with attributes such as its area, end-

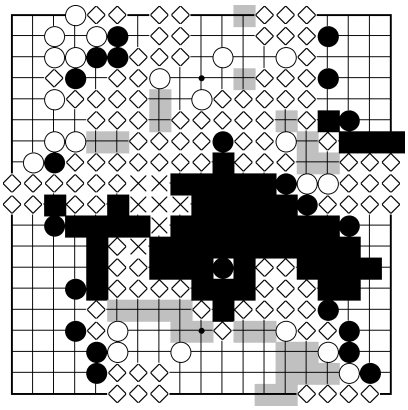

Figure 10: Near (shaded), junction ($\diamond$) and far away ($\times$) points.

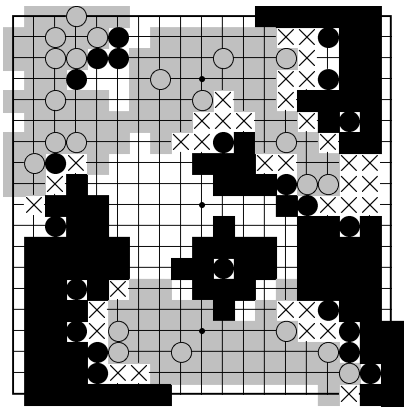

Figure 11: Connectible points for Black, White (shaded) and for both players ($\times$).

points, and status. The two endpoints of each divider are either stones of the divider's color, or the edge of the board. Figure 7 shows summary overviews of all points contained in some divider or potential divider, for Black and for White. The heuristic status of groups is used in this and future steps for filtering, to ignore dividers associated with dead groups of stones.

2. Separately for each color, *zones* and *potential zones* are computed next, as the result of a board partitioning process using non-dead stones, dividers and (only for potential zones) potential dividers of the same color. Using further heuristic information such as the connectability map shown in Figure 11, a *safety status* of each zone is computed: Figure 8 shows the *safe*, *potential*, and *threatened* zones for both players.

    - Safe zones are considered safe territory. They are surrounded only by stones and dividers, and the interior is strongly controlled by stones, as

measured by a heuristic.

- Potential zones are considered potential territory. They are weaker than safe zones, either because part of the boundary is only a potential divider, or because the control over the inside is not strong enough. An example is shown in the lower left corner in the left picture of Figure 8. Black's large area is surrounded by dividers and stones, but the current heuristic judges that the large inside area is too open to be safe, so the area is only potential territory. This is a somewhat pessimistic judgment.
- Zones that are even weaker than potential territory are evaluated as threatened. For example, a large area in the center right is loosely surrounded by black stones, but it has a large number of weaknesses.
- Zones that contain strong opponent groups are marked as *unused*, since the player has no scope for development there. An example would be the top right corner in Figure 8. This whole corner is surrounded on a large scale by white stones, divider and potential dividers. However, it contains a safe black corner group, so from White's point of view this zone is currently worthless.

Most points, except for zone boundaries, are part of both a black and a white zone.

3. Next, *divider-adjusted distances* from both Black and White stones are calculated, as shown in Figure 9. This distance function computes the Manhattan distance from each point to the nearest stone of the given color. It stops at opponent stones and dividers, and computes distances up to a given maximum, which is set to 5 in the current implementation.

4. Given distance information, points that are not territory or potential territory are classified as either *near* to one player, *junction points*, or as *far away* from both players. The result is shown in Figure 10.

4 Full-board Evaluation

Explorer uses Chinese rules by default. For full-board evaluation, the exactly and heuristically evaluated parts of the board are treated differently. For exactly evaluated parts, the mean values of each local position are computed and then added up. For safe territories and decided semeai, this value is identical to the size of the territory.

For the heuristically evaluated rest of the board, the number of points of each type is computed first. The contribution to the overall score is obtained by multiplying this number by a weight factor ranging from +1 for sure Black points to -1 for sure White points. The weights are currently set as follows:

1. safe territory: each point in a *safe territory* counts as +1 for Black, or -1 for White, towards the total score.

2. *potential territory*: each point counts as $\pm 1/2$.

3. Each *near* point is counted as ± 0.2.

4. *Junction points* and *far-away points* have a weight of zero, they do not contribute to the score.

The final score is the sum of the board score plus the komi. When using Japanese rules, adjustments are made for handicap stones.

5 Open Problems and Future Work

This section describes some ideas that a good evaluation function for Go should implement, but that are not yet included in Explorer.

5.1 Maximize the Chance of Winning, Not the Score

In point-scoring games such as Go it is natural to approximate the expected final score in the evaluation function. However, it is important to take the overall situation into account to maximize the chance of winning.

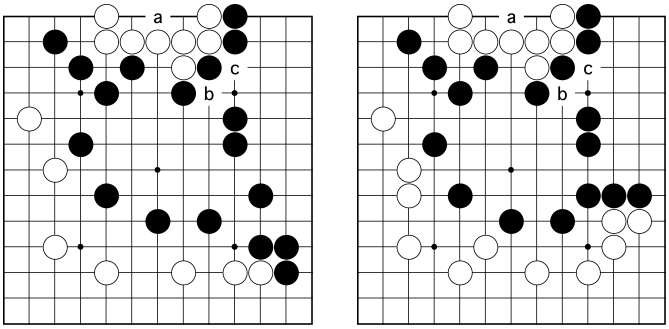

Figure 12: Maximizing the score vs maximizing the chance of winning.

Figure 12 shows two example positions. In both positions, Black can try to kill the group at the top with move *a*, an all-out attack that has a reasonable chance of success. Black's alternative is to play a defensive move at *b* and let White live. If Black plays *a*, White will cut at *c* and thrash around in Black's area, trying to make the group alive. If White succeeds, Black will suffer a big loss. Black's best strategy depends on an assessment of the overall position. On the left, Black can win easily and safely by playing the defensive move at *b*. On the right, Black is behind in territory, so the attack at *a* is the only chance to win. No matter how the unstable positions resulting after move *a* are evaluated, it is impossible for a score-maximizing program to find the correct strategy in both cases. It is essential to assess the winning probability instead.

The author believes that several Go programs, including Michael Reiss' *Go4++*, already implement similar ideas. Other programs include at least some simple related strategies, such as changing the weights of aggressive or defensive moves according to the score estimate.

5.2 Representing Evaluation Uncertainty

In games such as chess, the idea of quiescence search is essential for improving evaluation quality. Positions of high volatility should not be evaluated, but searched deeper. The same principle holds for Go. For example, the global search in David Fotland's program *The Many Faces of Go* mainly functions as a quiescence search: only moves that simplify the status of weak groups (save, kill) are considered in deeper stages of the search. Bruno Bouzy [4] has proposed to use fuzzy functions to represent evaluation uncertainty in Go.

5.3 Threat Evaluation

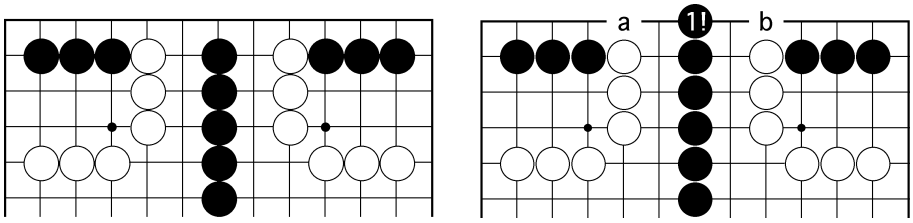

Figure 13: Proving the safety of blocks and territories.

The heuristic safety evaluation of territories is based on the assumption that the opponent's moves can be answered locally. Definitions of individual territories are usually not robust with regard to double threats. Even if two regions are each safe by themselves, there might be a move that threatens both at the same time. Figure 13 shows an example. Each corner, taken by itself, is a safe territory for White. The black stones inside can neither live nor connect to the outside. However, move 1 in the picture on the right is a double threat. Black will be able to connect one side, either at *a* or at *b*, and destroy one of the corner territories.

The author believes that several of the stronger Go programs perform at least a limited kind of threat analysis. However, there do not seem to be any publications about such methods.

5.4 Some Further Suggestions for Future Work on Evaluation in Go

Here is an (incomplete) list of further research topics in Go position evaluation.

- Develop methods to measure the quality of an evaluation function.
- For each game stage, develop a test set of full-board positions with detailed evaluations provided by human master players.
- Use territorial evaluation as a secondary criterion in goal-oriented searches. For example, a Life and Death solver could be modified to also maximize the territory of the group, or to capture a group on as large a scale as possible.

- Further automate mathematical endgame analysis techniques and their application to computer Go.
- Develop smooth transition functions between exact and heuristic evaluations, that can use a mix of both evaluation components.
- Generalize the methods for exact evaluation of territories, semeai and endgames.
- Modify Life and Death engines such as Thomas Wolf's *GoTools* to generate exact evaluations of local situations.
- Like double threats, ko fights are another case where normal territory definitions can be overturned. In order to win a ko, it may be necessary to allow the opponent two moves in a row. Many otherwise safe structures collapse under these circumstances. Concepts such as Benson's unconditional life, Popma and Allis' *X-Life* [16], and Tajima and Sanechika's *Possible Omission Number* [19] may form a basis to develop more refined evaluations here.

References

[1] D.B. Benson. Life in the game of Go. *Information Sciences*, 10:17–29, 1976. Reprinted in Computer Games, Levy, D.N.L. (Editor), Vol. II, pp. 203-213, Springer Verlag, New York 1988.

[2] E. Berlekamp. The economist's view of combinatorial games. In R. Nowakowski, editor, *Games of No Chance: Combinatorial Games at MSRI*, pages 365–405. Cambridge University Press, 1996.

[3] B. Bouzy. *Modélisation cognitive du joueur de Go*. PhD thesis, Université Paris 6, 1995.

[4] B. Bouzy. There are no winning moves except the last. In *Proceedings IPMU*, 1996.

[5] B. Bouzy and T. Cazenave. Computer Go: An AI-oriented survey. *Artificial Intelligence*, 132(1):39–103, 2001.

[6] J. Burmeister and J. Wiles. AI techniques used in computer Go. In *Fourth Conference of the Australasian Cognitive Science Society*, Newcastle, 1997.

[7] K. Chen. Some practical techniques for global search in Go. *ICGA Journal*, 23(2):67–74, 2000.

[8] B. Fraser. *Computer-Assisted Thermographic Analysis of Go Endgames*. PhD thesis, University of California at Berkeley, forthcoming.

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

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

[11] M. Müller. Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames. In *IJCAI-99*, pages 578–583, 1999.

[12] M. Müller. Race to capture: Analyzing semeai in Go. In *Game Programming Workshop in Japan '99*, volume 99(14) of *IPSJ Symposium Series*, pages 61–68, 1999.

[13] M. Müller. Generalized thermography: A new approach to evaluation in computer Go. In J. van den Herik and H. Iida, editors, *Games in AI Research*, pages 203–219, Maastricht, 2000. Universiteit Maastricht.

[14] M. Müller. Partial order bounding: A new approach to evaluation in game tree search. *Artificial Intelligence*, 129(1-2):279–311, 2001.

[15] M. Müller. Computer Go. *Artificial Intelligence*, 134(1–2):145–179, 2002.

[16] R. Popma and L.V. Allis. Life and death refined. In H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3*, pages 157–164. Ellis Horwood, 1992.

[17] W. Spight. Extended thermography for multiple kos in Go. In H.J. van den Herik and H. Iida, editors, *Computers and Games. Proceedings CG'98*, number 1558 in Lecture Notes in Computer Science, pages 232–251. Springer Verlag, 1999.

[18] W. Spight. Go thermography - the 4/21/98 Jiang-Rui endgame. In R. Nowakowski, editor, *More Games of No Chance*. Cambridge University Press, 2002.

[19] M. Tajima and N. Sanechika. Estimating the Possible Omission Number for groups in Go by the number of n-th dame. In H.J.van den Herik and H.Iida, editors, *Computers and Games: Proceedings CG'98*, number 1558 in Lecture Notes in Computer Science, pages 265–281. Springer Verlag, 1999.

[20] B. Wilcox. Computer Go. In D.N.L. Levy, editor, *Computer Games*, volume 2, pages 94–135. Springer-Verlag, 1988.

[21] S.-J. Yan, W.-J. Chen, and S.C. Hsu. Design and implementation of a heuristic beginning game system for computer Go. In *JCIS'98*, volume 1, pages 381–384, 1998.

[22] S.-J. Yan and S.C. Hsu. A positional judgment system for computer Go. In H.J. van den Herik and B. Monien, editors, *Advances in Computer Games 9*, pages 313–326, Maastricht and Paderborn, 2001.