SOLVING GO FOR RECTANGULAR BOARDS

Erik C.D. van der Werf[1] *Mark H.M. Winands*[2]

Eindhoven, The Netherlands Maastricht, The Netherlands

ABSTRACT

In 2003, the solution for the 5×5 Go board was published in this journal. The current article presents the game-theoretic values for rectangular boards up to a surface of 30 intersections under Chinese rules. The result was achieved by improving the $\alpha\beta$-based solver MIGOS. Moreover, the article identifies errors in published human solutions by comparing them with our computer solutions.

1. INTRODUCTION

Already for fifty years, building strong game-playing programs is one of the main targets of AI researchers. The principal aim is to witness the "intelligence" of computers. In the last twenty years a second aim has emerged: establishing the *game-theoretic value* of a game (van den Herik, Uiterwijk, and van Rijswijck, 2002). The game-theoretic value indicates whether a game is won, lost, or drawn when both players play optimally. In Go it also provides the amount of points by which a game is won (or lost), which is important because the second player normally receives komi points to compensate the advantage of the initiative (Uiterwijk and van den Herik, 2000). For solving a game, three "layered" definitions exist (Allis, 1994).

Ultra-weakly solved. For the initial position, the game-theoretic value has been determined. The game of Hex is an instance of an ultra-weakly solved game (Nash, 1952).

Weakly solved. For the initial position, a strategy has been determined to obtain at least the game-theoretic value of the game. The following games were weakly solved: Go-Moku, (Allis, van den Herik, and Huntjens, 1996), Nine Men's Morris (Gasser, 1996), Renju (Wágner and Virág, 2001), Checkers (Schaeffer *et al.*, 2007), and Fanorona (Schadd *et al.*, 2008).

Strongly solved. For all legal positions, a strategy has been determined to obtain the game-theoretic value of the position. Examples are Kalah (Irving, Donkers, and Uiterwijk, 2000), Awari (Romein and Bal, 2003), and Connect-Four (Tromp, 2008).

When solving a game is intractable with current hardware means, researchers have tried solving the smaller variants. For instance, the 8×8 variant of Hex (Henderson, Arneson, and Hayward, 2009) and the 6×6 variant of LOA (Winands, 2008a) have been solved recently. Solving smaller variants is relevant for the following three reasons. (1) Solving smaller variants may function as a pre-study for solving the regular variant. (2) The smaller variants are studied by humans who are curious for the results (e.g., as for 5×5 Go, Davies, 1994). (3) The results can be used for subgames when playing a game at the regular board size (Müller, 1995).

Solving small Go boards has also been a popular topic of research. The earliest computer analysis for Go on smaller boards were reported by Thorp and Walden (1972). The largest boards investigated by them were 1×5, 2×4, and 3×4. In the following years more computer proofs were established (e.g., 4×4 by Sei and Kawashima, 2000). Up to now, the largest *square* board for which a computer proof has been published is 5×5 by van der Werf, van den Herik, and Uiterwijk (2003). It is a full-board win for the first player (Black).

[1]Eindhoven, The Netherlands. Email: erikvanderwerf@gmail.com

[2]Games and AI Group, Department of Knowledge Engineering, Faculty of Humanities and Sciences, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands. Email: m.winands@maastrichtuniversity.nl

For the bigger rectangular boards (e.g., 4×7, 3×8, 5×6), the published (numeric) results are human solutions. A complication is that the rule set used is not consistent and sometimes even unspecified. For some boards Chinese rules are used, for others New Zealand or American rules are applied. But the biggest problem is that these solutions, as for instance the ones reported in van den Herik *et al.* (2002) and Drange (2002), contain several errors irrespective of the rules used.

This article presents a search-based approach of weakly solving rectangular boards up to a surface of 30 intersections. The search method is the well-known $\alpha\beta$ framework extended with several domain-dependent and domain-independent search enhancements. The program is called MIGOS II, an improved version of the one that solved 5×5 Go. Moreover, in this article we perform together with Ted Drange (2009) a detailed comparison with a selected number of human solutions.

The article is organized as follows. Section 2 briefly provides details of the rule set used. Section 3 describes the new enhancements in MIGOS. The results of rectangular boards with sizes up to 30 intersections are given in Section 4. Subsequently, Section 5 identifies the errors in the human solutions by comparing them with the computer solutions. Finally, Section 6 gives conclusions and an outlook.

2. GO RULES

This section briefly describes the rules used in the program MIGOS. It is beyond the scope of this article to explain all rules in detail. For a more elaborate introduction we refer to van der Werf (2004).

Many rule sets exist for the game of Go (e.g., Chinese, Japanese, American, SST (Ing), New Zealand). Although all major rule sets agree on the same general idea of how the game is to be played, there exist several subtle differences. These differences mainly deal with repetition (the ko rule), life and death, suicide, and the scoring method at the end of the game.

The primary difference between rule sets is the scoring method. The two main scoring methods are: (1) area scoring, which counts intersections on the board that are occupied or controlled by one colour, and (2) territory scoring, which counts intersections surrounded by living stones plus prisoners (dead/captured opponent stones). In practice, it is rare to observe more than a one-point difference between scoring methods. However, the typical one-point difference may appear in roughly 50% of the games.

The scores under Japanese rules (using territory scoring) come from a fine-grained distribution (taking steps of one point). The scores under Chinese rules (using area scoring) are more sparse (they are peaked in steps of two points because one-point steps under area scoring require neutral intersections, which are rare). As a consequence one may argue that the Japanese rules are slightly more interesting than the Chinese rules. However, it is well known that Japanese rules are extremely difficult (and by some even considered impossible) to implement in a program due to ambiguities and inconsistencies in the official texts. Chinese rules also suffer from some ambiguity, but to a much lesser extent. Therefore, it is the natural choice for Go programmers to prefer Chinese rules.

The rules implemented in the program MIGOS are a formalization of the official Chinese rules (Davies, 2001). Most notably, the game rules in MIGOS have a long-cycle rule that is consistent with all the ad-hoc examples presented in Chapter 3 (“Rules for the Referee”) section 20 (“Reappearance of the same board position”) of the Chinese rules. For details about the MIGOS rules the reader is referred to van der Werf (2004).

Several rule details are configurable in MIGOS. In nearly all cases these details do not affect the game-theoretic value of the empty board. However, in case of doubt, we always try different configurations to ensure that the first player score is not affected. If the score is affected we will report it and its cause in this article.

3. MIGOS II

In this section we describe MIGOS II, the program that solves rectangular Go boards with different dimensions. First, in Subsection 3.1 we briefly give an overview of MIGOS 2003, the program that serves as a foundation for the current solver. Next, in Subsection 3.2 we explain the enhancements made to the evaluation function. Subsequently, Subsection 3.3 presents enhancements regarding the transposition table, and discusses potential problems caused by hash collisions. Finally, other (possible) improvements are discussed in Subsection 3.4.

3.1 MIGOS 2003

The program MIGOS (MIni GO Solver) performs an $\alpha\beta$ depth-first iterative-deepening search in the PVS framework (Marsland, 1986). A transposition table (Nelson, 1985) is applied to prune subtrees, narrow the $\alpha\beta$ window, and improve move ordering. At all interior nodes that are more than 3 ply away from the leaves, it generates all moves to perform Enhanced Transposition Cutoffs (Schaeffer and Plaat, 1996). The effect of using a transposition table is further enhanced by looking for symmetrical positions that have already been searched. On rectangular boards these symmetries typically reduce the state space by a factor approaching 4 (non-square board dimensions break one symmetry, and the colour symmetry is not used).

MIGOS is equipped with a method for early detection of sure bounds on the final score by recognizing unconditional life (Benson, 1976; Müller, 1997) and unconditional territory (van der Werf *et al.*, 2003). It is not only used at leaf nodes but also at internal nodes in order to improve the efficiency of the search. Lower and upper bounds can be computed that either generate a direct cutoff or narrow the alpha-beta window. Unconditional territory is further used for reducing the branching factor. By taking care of a few exceptions, moves that reside inside unconditional territory are not examined since they cannot change the outcome of the game.

Subsequently, the move ordering of MIGOS works as follows: first the transposition move, second the first move sorted by the history heuristic (Schaeffer, 1983), third the first killer move (Akl and Newborn, 1977), fourth the second move sorted by the history heuristic, fifth the second killer move, and finally the remainder of the moves sorted by the history heuristic. In positions that are searched sufficiently deep (at least 5 plies away from the leaves) the move ordering is interleaved with sibling promotion (Dyer, 1995).

Finally, we remark that MIGOS is entirely written in C. More information about MIGOS' search and evaluation function for scoring Go positions can be found in van der Werf (2004).

3.2 Evaluation

Two minor changes were made to the evaluation function. The first change is that at the end of the game stones with solely one liberty are only considered dead (and consequently removed before counting the final score) if the opponent does not have such stones. This change prevents the program from returning a non-conventional score in final positions where both sides have stones that can be captured in one move; even though either side would lose from making the actual capture.[3] MIGOS 2003 removed all stones with one liberty before counting the score. As before, the program can also be configured to treat all stones as alive. However, we only use this option in case of doubt because it is rather inefficient.

The second change is that MIGOS II (optionally) distinguishes between normal scores (obtained from the ordinary counting procedure after consecutive passes) and exceptional scores (used to assign a value for termination by long-cycle repetition). In practice, the most important application of this change is to distinguish long-cycle ties from ordinary ties. This facilitates analysis because we can, e.g., configure White to win in the case of an ordinary tie (where both sides control an equal number of board points) while Black wins in case of a balanced long-cycle repetition.

3.3 Transposition Table

The transposition table implementation was improved in a number of ways. Most notably, MIGOS' original TWODEEP replacement scheme (cf. Breuker, Uiterwijk, and van den Herik, 1996) was replaced by a sequential multi-probe replacement scheme similar to the one suggested by Beal and Smith (1996). Experiments in MIGOS confirmed their results that the sequential multi-probe replacement scheme is more efficient than TWODEEP. Our multi-probe scheme searches four consecutive entries and always replaces the one representing the shallowest sub-tree. This is slightly different from Beal's implementation which only replaces an entry if the new result has a higher priority (deeper sub-tree). Our implementation of always storing the new results turns out to be more efficient. This is probably because it improves the efficiency of re-searches (which occur frequently in PVS). Moreover, we have tested searching more than four consecutive entries, and have confirmed Beal's result that

[3] An example of such a position is the extremely rare hanezeki.

this reduces the node counts even further. However, due to a slow-down of our search in nodes-per-second, the reduction in node-counts usually does not provide a significant effective speedup.

Other changes include setting some flags to make more efficient use of previous proofs, using packed structures to increase the maximum transposition table size, and a slightly more efficient use of Enhanced Transposition Cutoffs (cf. Schaeffer and Plaat, 1996). Below we provide some details on preventing transposition table problems.

3.3.1 Preventing Transposition Table Problems

The transposition table implemented in MIGOS II uses Zobrist hashing (Zobrist, 1970) to identify board situations. Problems may occur when two distinctly different board situations are mapped to the same hash value, which is commonly referred to as a hash collision.

There are two problems causing potential hash collisions. The first is a fundamental limitation of hashing, the second is implementation dependent and lies in the willful omission of potentially relevant properties of the game state from the hash. Both will be discussed in the following paragraphs.

The fundamental problem is that the hash may be imperfect, i.e., when the number of properties potentially included in the hash is larger than the number of available bits some information may be lost. Consequently, there may be a small but non-zero probability that two investigated states are mapped to the same hash value. The probability of such an event may be estimated using the following formula

$$P(errors) \approx \frac{M^2}{2N} \tag{1}$$

where M is the number of unique positions and N the number of possible hash values, assuming M is sufficiently large and small compared to N (Breuker, 1998). For a search of 500G unique nodes with a Zobrist hash of effectively 88 bits (26 address, 64 lock, 2 bits lost to sequential multi-probe search), the probability of at least one error is 4×10^{-4}, which is acceptable for our purposes. Nevertheless, as an extra precaution MIGOS II also tests the legality of moves proposed by the transposition table. Consequently, if such an error would ever start to occur frequently it would show up in the logs (none were observed in our experiments). To overcome the problem altogether one could also try to store the complete game state. However, since this is only feasible under some rules settings and does not scale up to larger boards it is not used in MIGOS II.

The second problem is that collisions may also occur when relevant properties are omitted from the hash value. Problems typically arise when history is ignored. Depending on the history, a board situation may have a certain value. If this information is ignored a node may receive an incorrect value. This is called the graph-history interaction problem (GHI) (Campbell, 1985; Kishimoto and Müller, 2005). The relevant properties to include in the hash are determined by the specific Go rules used. It is generally considered impractical to include the complete game history (i.e., all previous board positions), even when the rules dictate that it may be relevant (i.e., superko rules). Most Go programmers therefore choose to omit this information, and accept the rare errors that this may cause. Although MIGOS II can be compiled to account for previous boards in the hash, it is generally not used because it is too inefficient. Fortunately, all this is much less a problem under traditional Go rules.

In the design of MIGOS II and its preferred rule set considerable care was taken to make it feasible to include all relevant properties of a board situation in the hash, which happens to coincide well with traditional rules. Consequently, correctness can be guaranteed at the expense of only a relatively minor performance penalty (compared to simply ignoring properties that are unlikely to make a difference).

Situational properties typically included in the hash of MIGOS II are:

- player to move
- basic ko
- number of consecutive passes
- last play was basic ko capture (before passing started)
- last two plays were basic ko captures
- number of Black/White captures
- number of Black/White passes (not necessarily consecutive)

A drawback of including all potentially relevant situational information in the hash is that it reduces the ability to generalize over previously encountered positions that are nearly identical (i.e., same configuration of stones on the board, same player to move, same set of legal moves), thus reducing search efficiency. Consequently, for analysis we may at times choose to omit some properties just to obtain initial results faster.

Probably the most significant change, compared to MIGOS 2003, is that the option to include the numbers of captures and passes in the hash is now used less frequently (it was originally introduced to solve the problem of depth-shifted transpositions and distinguish situations in cycles, see van der Werf (2004) for details). Instead the default setting is now to accept depth-shifted transpositions for heuristic results (which occasionally enables the search to look beyond the horizon). Depth-shifted proofs (originating further from the root) are still not accepted. However, instead of completely discarding such proofs they are now converted to extreme heuristic results (their values are adjusted just enough to move them into the heuristic range), forcing the search to verify while still providing useful information for move ordering. Other minor changes regarding the situational properties in the hash mainly deal with optional settings to experiment with different game-end criteria.

Although most potential GHI problems are solved in MIGOS II one potential problem remains. Since superko is not used, balanced long-cycle repetition (where both sides capture an equal number of stones in each cycle) is scored as a long-cycle-tie. Normally, there is no need to distinguishing long-cycle-ties from heuristic scores, so MIGOS II simply assigns a value in the heuristic range (typically 0). However, when optimal play leads to a long-cycle-tie this setting prevents the search from returning a proof. Alternatively, the long-cycle-tie can be assigned a value corresponding to a minimal proven win, for either Black or White, but this then becomes prone to the ordinary GHI problems normally observed only under superko rules (so occasionally some additional situational properties have to be included in the hash, which reduces efficiency).

3.4 Other (Possible) Improvements

MIGOS II is equipped with two (safe) forward pruning methods, null move (cf. Donninger, 1993) and late-move reductions (cf. Romstad, 2006). Although they were sometimes able to find a proof faster, most of the time several extra plies had to be searched to find the proof due to a missed line. It was unclear whether forward pruning would decrease the search effort. They were therefore not used for the experiments.

4. RESULTS FOR RECTANGULAR BOARDS

MIGOS II was used to compute the values for the rectangular boards up to a surface of 30 intersections. Initially, several different machines were used to solve various boards in parallel. To make the results comparable, and for verification, the experiments were repeated on an Intel Core2 3GHz (single thread) machine using a transposition table of 0.94 Gb. (2^{26} entries, 15 bytes per entry). Only for the largest boards the experiments were not repeated on this default machine because that would have been too time consuming; these results are therefore marked by the comment non-default. In Table 1 the results are given together with their search depth, nodes investigated,[4] and best move for the initial position.

All boards were first searched with a fractional komi just below the expected game-theoretic value, which is equivalent to searching with the expected game-theoretic value as komi and assigning ties to the first player (finding a first-player win then proves a lower bound on the score). We remark that in MIGOS II searching close to the correct komi is generally more efficient than simply searching with komi at zero because the heuristic range of the evaluations is used more effectively. Boards that did not get a maximal score (e.g., 4×6), could in principle contain an undetected deep variant that might raise the score further. To rule out the possibility of a higher score such boards were searched again with komi just above the expected game-theoretic value (equivalent to searching with integer komi and assigning ties to the second player). Finding the second-player win then established both the lower and the upper bound on the score, thus confirming that they are indeed correct. So, when two values are shown in Table 1 for depth/nodes/time, the first and second refer to the lower and upper bound proofs, respectively. For full-board wins only one value is shown because a single proof suffices.

[4] In iterative deepening PVS the same node is usually investigated multiple times; we count each visit. The reported node-counts are therefore upper bounds on the size of the minimal proof-graph.

Surface	Board	Result	Depth	Nodes	Time	First move	comments
30	3×10	≥6	29–	1.31T–	30.8(d)–	central	non-default [5]
30	5×6	4	27–38	214G–	4.5(d) –	central	non-default [6]
28	4×7	28	37	363G	10.6(d)	central	non-default [7]
27	3×9	5	29–30	289–294G	4.3(d)	centre	non-default
25	5×5	25	23	603M	14.3(m)	centre	
24	3×8	24	37	120G	1.86(d)	central	
24	4×6	8	29–34	3.13–13.2G	1.26–5.41(h)	central	
22	2×11	6	31–24	281G–18.3G	7.77(d)–12.2(h)	central	non-default
21	3×7	21	39	356M	7.87(m)	centre	
20	2×10	4	23–24	725–942M	14.1–25.9(m)	central	
20	4×5	20	21	20.3M	29.6(s)	central	
18	2×9	18	33	136M	2.76(m)	central	
18	3×6	18	17	2.74M	3.22(s)	central	
16	2×8	16	21	9.92M	10.2(s)	central	
16	4×4	2	21–23	695–826k	1.09–1.73(s)	central	
15	3×5	15	13	274k	276(ms)	centre	
14	2×7	14	15	705k	576(ms)	central	
12	1×12	2	53–48	1.07G–679M	18.7–11.5(m)	2,4	[8]
12	2×6	12	13	89.5k	66.2(ms)	central	
12	3×4	4	13–23	40.4–85.8k	41.4–86.4(ms)	central	
11	1×11	2	27–39	534k–9.11M	538(ms)–5.39(s)	2,4,6	
10	1×10	1	28–28	256–360k	224–272(ms)	2	
10	2×5	10	9	7.32k	5.55(ms)	central	
9	1×9	0	17–24	32.8–58.9k	23.7–43.3(ms)	2,4,5	
9	3×3	9	11	2.61k	2.13(ms)	centre	
8	1×8	3	15–30	10.6–57.1k	7.45–33.8(ms)	2	
8	2×4	8	9	2.05k	1.37(ms)	central	
7	1×7	2	9–22	1.97–5.84k	2.19–4.78(ms)	2,4	
6	1×6	1	9–18	1.09–2.12k	0.949–2.06(ms)	2	
6	2×3	0	16–25	1.57–5.88k	1.57–6.89(ms)	central	[9]
5	1×5	0	9–8	410–450	0.55–0.54(ms)	2,3	
4	1×4	4	7	131	0.327(ms)	central	
4	2×2	0	10–11	391–350	0.59–0.90(ms)	any	[10]
3	1×3	3	1	6	0.119(ms)	centre	
2	1×2	0	8–8	128–81	0.462(ms)	any	

Table 1: Detailed results.

In most experiments the game-theoretic values were already known from earlier experiments. However, in case the game-theoretic value is unknown one may simply start at 0; then when a lower bound proof is found one starts a search for the upper bound proof. In case this second search fails one automatically obtains a new lower bound proof and the process repeats until either a full-board win or an upper bound proof is found.

Results for $1\times n$ boards are included for completeness. However, due to some design constraints MIGOS II is rather inefficient on $1\times n$ boards. Consequently, comparison of search times with results on 2-dimensional boards is not straightforward. Therefore, we stopped computing $1\times n$ boards after the 1×12 board. Similar inefficiencies also play a role on $2\times n$ boards when n becomes greater than 10. Here we stopped after the 2×11 board.

All $1\times n$ boards were analyzed with a rules setting that when the game ends (after consecutive passes) all stones are assumed alive. This was done as a precaution because long-cycles occur frequently for $1\times n$ boards. Other-

[5] Lower bound proof only, 2 months of search on a single CPU did not suffice to return an upper bound proof.
[6] Lower bound proof only, upper bound proof was manually distributed over multiple machines.
[7] First move selected manually to save time.
[8] Needs extra precautions against GHI. We ran into a rather extreme example on 1×12 with a long cycle of 38 ply occurring at depth 54!
[9] Sensitive to rules details, needed additional precautions against GHI.
[10] Sensitive to rules details. The result remains a tie as long as komi is in the range [-1,1] because either side can force a balanced long cycle.

wise it may happen that a capturable block becomes uncapturable because the capture might cause an undesired long-cycle result. For other boards the reported results are "for the more efficient rules setting" that in a terminal position all blocks with one liberty are assumed dead, unless when both sides have such block(s). We verified that this setting had no impact on the reported first-player scores.

Table 1 confirms that without komi White never wins (which can easily be proved theoretically using a strategy stealing argument). There are only ties for some instances of the $1\times n$ and $2\times n$ boards. In all other cases Black wins. Moreover, we see that full-board wins not only occur for the smaller boards, but also for some bigger ones (e.g., 3×8 and 4×7).

At equal surfaces, elongated boards are generally more difficult to solve than boards that are closer to a square shape. Moreover, we observe that for the odd-surface boards the unique *centre* is usually the best opening move. On even-surface boards, where there is no unique centre, the best opening move is on any of the 2 or 4 *central* intersections. The only exception (so far), where the centre is inferior, is for some $1\times n$ boards (the best opening move on $1\times n$ normally takes the second point to secure one end).

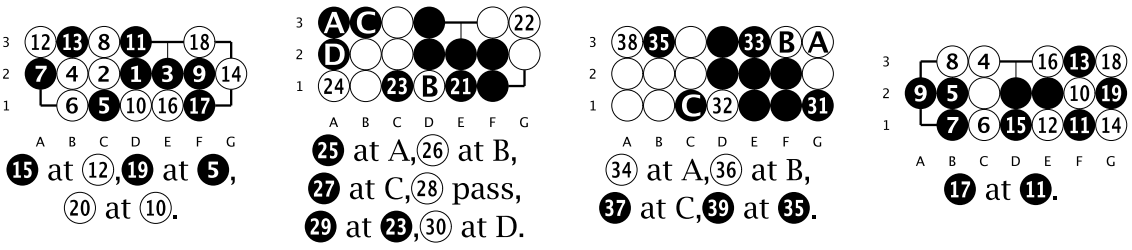
Table 1 shows us that by using one CPU most boards could be solved in approximately a week. Except for 5×6, multiple CPUs had to be employed to establish the upper bound proof. Finally, we remark that 5×5, which took 2.7 hours to be solved in 2003 (cf. van der Werf *et al.*, 2003), is nowadays cracked in less than 15 minutes. This is not only thanks to new hardware. Due to improvements of the search engine the program investigates roughly 40% fewer nodes (when using identical komi, rules settings, and memory footprint).

5. SURPRISING RESULTS

For several boards we found different results from the solutions reported by Drange (2002) and van den Herik *et al.* (2002). After some analysis over email between the first author and Ted Drange (2009) the human errors were identified and our results confirmed. This section gives an overview.

5.1 3×7

The 3×7 board was believed to be a win by 5 points (Drange, 2002; van den Herik *et al.*, 2002), but turns out to be a win by 21 points. The optimal line of play, leading to a full-board win, is presented in Figure 1. Diagrams a–c show one of the deepest variations. White has many alternatives, such as playing move 4 at **c3** as shown in Diagram d. However, without a Black mistake there is no variation that allows White to live on 3×7.



(a) Moves 1–20. (b) Moves 21–30. (c) Moves 31–39. (d) Variation.

Figure 1: Optimal play on 3×7.

Depending on the rule set Black may be tricked into a premature game-end. If the game ends by two consecutive passes, or by superko rules, Black should be careful not to play 31 at **a3** or **b3** because that would enable White to make moonshine life (White captures, Black has to pass, White passes as well and Black cannot recapture the ko). If Black simply plays 31 as depicted in Figure 1c he[11] avoids all difficulties.

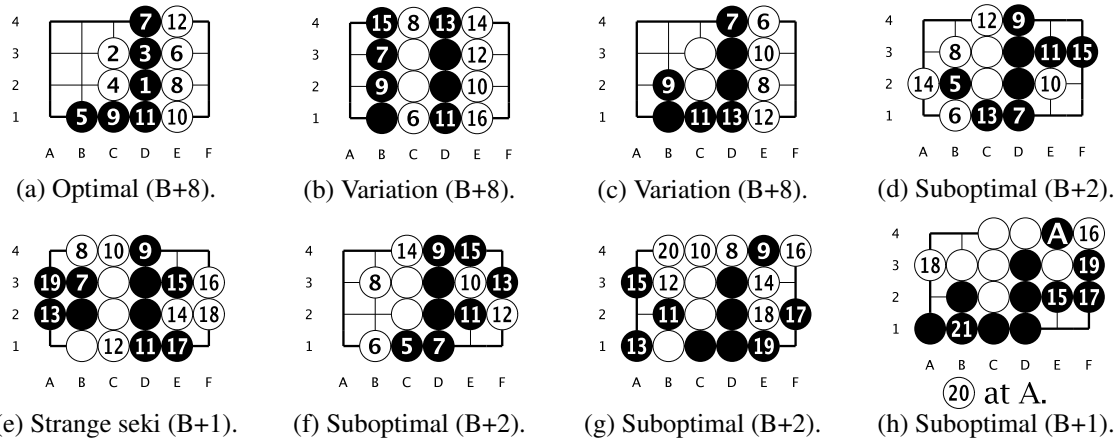
The main problem with the human analysis of 3×7 was that it was simply not studied deep enough. To some extent this is understandable. MIGOS II also finds the 5 point win quite quickly at depth 23. However, to obtain more than 5 points requires another 16 ply of search with no intermediate improvements until the full board win is found at depth 39 (which is remarkably deep for a board with only 21 intersections).

[11] For brevity, we use 'he' and 'his' when 'he or she' and 'his or her' are meant.

5.2 4×6

The 4×6 board believed to be a win by 1 point (Drange, 2002; van den Herik *et al.*, 2002) turns out to be a win by 8 points. Three variations for optimal play are shown in Figures 2a to 2c. Human analysis missed the key move 5 at **b1** in Figure 2a.

There are a few interesting suboptimal lines as well. For example, if Black plays move 5 at **b2** or **c1**, he only achieves a 2-point win, as shown in Figures 2d, f, and g. If Black plays 15 at 18 in Figure 2g, he achieves the even more inferior 1-point result as depicted in Figure 2h.



(a) Optimal (B+8). (b) Variation (B+8). (c) Variation (B+8). (d) Suboptimal (B+2).

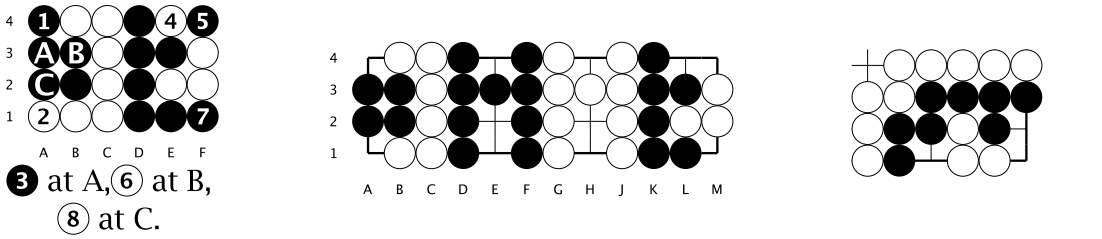
(e) Strange seki (B+1). (f) Suboptimal (B+2). (g) Suboptimal (B+2). (h) Suboptimal (B+1).

Figure 2: 4×6 variations.

5.2.1 Raising the Dead

If Black plays suboptimal twice with move 5 at **b2** and move 7 at **b3**, then the final position becomes a very special seki as shown in Figure 2e.

The position is seki because if Black would play **a4** (or **a1**) – in order to attempt to capture the White group with one eye – play continues as shown in Figure 3a. After Black throws in at A, to keep the white group at one eye, White starts a ko with move 4. Black has to respond with move 5 and White has now a ko-threat at B that Black must ignore (or he will lose his right group and as consequence the full board). After move 8 at C both sides have 12 points and the result is a tie, which is one point worse for Black than the original seki position.



(a) Why seki? (b) Global seki. (c) Irremovable potential ko.

Figure 3: Seki positions.

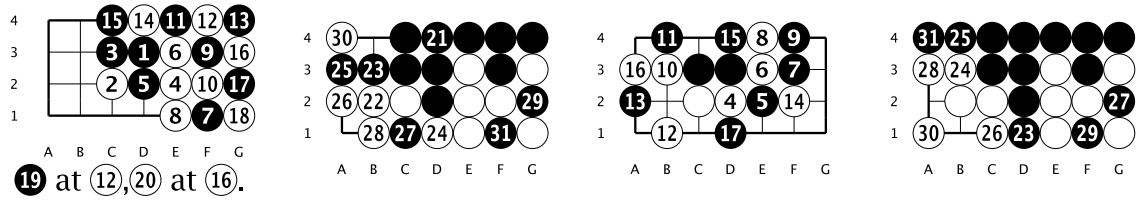
Black also cannot start the ko himself (he has no ko-threats) or play **e4** (he would lose the capturing race). White has to be careful not to play 20 **f4** (which would guarantee a local seki by removing the potential for ko) because in this position it would kill the surrounding white group and hence give the full board to Black.

It is interesting that the seki would remain even when the configurations of the left half and the right half of the board are non-adjacent as shown in Figure 3b. Generalizing further; if a configuration such as shown in Figure 3c would be present in any final position (after both players pass) White could claim life in seki for a variety of

one-eyed groups that would normally be dead inside the opponent's territory. Whether such a claim would be granted under traditional rules that attempt to localize shapes in the scoring procedure is doubtful. However, it is clear that this position is an interesting phenomenon, and to our knowledge may even be a new discovery.

5.3 4×7

The 4×7 board was believed to be a win by 4 points (Drange, 2002), but is a full-board win of 28 points. The optimal play for 4×7 is shown in Figures 4a and b. The key move is 7; it is rather similar to the key move on 4×6. Human analysis missed this move for the 4×7 board as well. If Black plays move 7 anywhere else, such as **e1**, **f2**, **f3**, **e4**, or even on the opposite side at **f4**, then White can live. White can play many other variations, such as the one shown in Figures 4c, but this generally leads to an even quicker loss. Black does not have much choice on 4×7. Most deviations directly give White the opportunity to live. An example of such a mistake is shown in Figure 4d where splitting White's group costs Black 16 points.

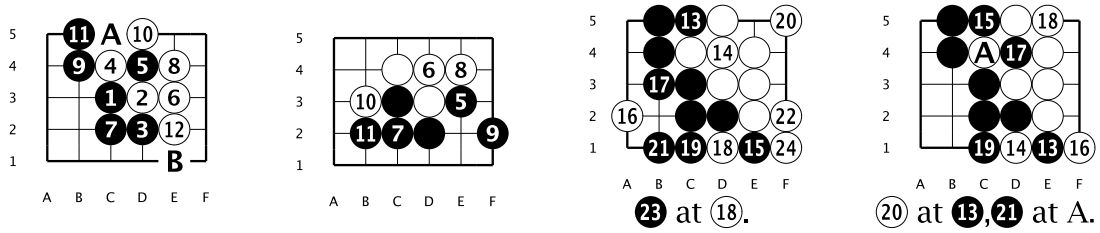


(a) Optimal 1–20 (B+28). (b) Continuation. (c) Simpler variation. (d) Mistake at 23.

Figure 4: 4×7 variations.

5.4 5×6

Human analysis in the literature did not agree on what the game-theoretic value was for the 5×6 board. Van den Herik *et al.* (2002) report it as a tie whereas Drange (2002) was convinced that it was a win by 2 points. Our computational proof reveals that this board is a win by 4 points (Black obtains 17 points and White 13 points). Figure 5a shows the first 12 moves. Playing move 3 at 4, or 7 at 9, would be a mistake. The keys to the 4-point win are moves 5 and 7.



(a) Moves 1–12. (b) Suboptimal. (c) Continuation A. (d) Continuation B.

Figure 5: 5×6.

Human analysis discarded playing move 5 at **d4**, most likely because they missed the key move at 7, and focussed mainly on the (suboptimal) continuation as shown in Figure 5b. If Black plays as in Figure 5b, at move 12 White can choose **a3**, **a4**, or **b4** which all lead to at a win by only 2 points for Black. One might think that by playing **a4**, and following a mirror strategy, the game could even become a tie. However, Black can force White to break symmetry and win by 2 points.

After the first 12 moves (as depicted in Figure 5a) Black can choose to play either at A or at B. Possible continuations are shown in Figures 5c and 5d, both leading to a 4-point win. For continuation A, White can also play 14 at 15 or 18, which leads to an equivalent result. If Black selects continuation B, White can answer passively at 15. However, if White plays the atari at 14 Black has to be careful not to capture directly, which would cost him 2 points.

6. CONCLUSIONS AND OUTLOOK

The main result of this article is that most rectangular Go boards up to a surface of 30 intersections have been (weakly) solved. This result was achieved by using an improved version of the Go solver MIGOS, called MIGOS II. An overview of the game-theoretic results is given in Table 2. The table indicates that, except for a few instances of $1\times n$ and $2\times n$ boards, all are a first-player win. Moreover, the computer proofs identified several errors in human solutions. The noticeable differences were unexpected full-board wins for 3×7 and 4×7, an 8-point win for 4×6, and a 4-point win for 5×6.

$m\backslash n$	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	3	4	0	1	2	3	0	1	2	2
2	0	0	0	8	10	12	14	16	18	4	6	
3	3	0	9	4	15	18	21	24	5	$\geq$6		
4	4	8	4	2	20	8	28					
5	0	10	15	20	25	4						

Table 2: First player scores on $m \times n$ boards.

The question now is when would 6×6 be solved? In 6 years time we went from a surface of 25 (i.e., 5×5) to a surface of 30 (i.e., 5×6) of solved Go boards. This was not only because of better hardware but also due to a better search engine.

We can try to predict when MIGOS II would be able to solve 6×6 in a reasonable amount of time by extrapolating the current results. Based on the results of Table 1, Figures 6 and 7 show how the complexity in terms of nodes and time respectively scale with the surface of the board.

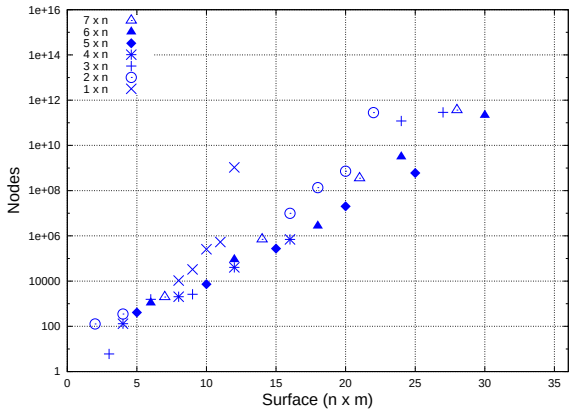


Figure 6: Board surface vs Nodes.

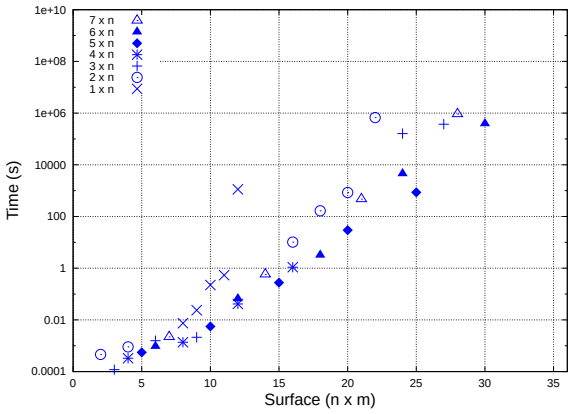


Figure 7: Board surface vs Time.

An optimistic extrapolation suggests that on current hardware MIGOS II would require a few years to solve 6×6. However, we could easily be underestimating by a factor of 100. Nevertheless, we believe that with some effort $\alpha\beta$-based solvers, such as MIGOS II, should be able to solve 6×6 within the next 5 years, especially because significant improvements in the evaluation function are still possible.

Other approaches are possible as well. Already now, modern Go programs using MCTS (Coulom, 2007) appear to play rather close to optimal on small boards such as 6×6 and 7×7. Even on 9×9 they are challenging opponents for professional players. MCTS programs play really strong due to heuristics. However, for solving a board this does not suffice because one has to deal with a voluminous proof tree that (at least for one side) includes even the most unlikely lines of play. Integrating minimax proving capabilities into MCTS programs as done in MCTS-Solver (Winands, Björnsson, and Saito, 2008b) appears to be a promising alternative strategy.

Acknowledgements

We are grateful to Ted Drange for his analysis, and to Jahn-Takeshi Saito and Peter Geurtz for their technical assistance.

7. REFERENCES

Akl, S. G. and Newborn, M. M. (1977). The principal continuation and the killer heuristic. *1977 ACM Annual Conference Proceedings*, pp. 466–473, ACM Press, New York, NY.

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence.* Ph.D. thesis, Rijksuniversiteit Limburg, Maastricht, The Netherlands.

Allis, L. V., Herik, H. J. van den, and Huntjens, M. P. H. (1996). Go-Moku Solved by New Search Techniques. *Computational Intelligence*, Vol. 12, No. 1, pp. 7–24.

Beal, D. F. and Smith, M. C. (1996). Multiple probes of transposition tables. *ICCA Journal*, Vol. 19, No. 4, pp. 227–233.

Benson, D. B. (1976). Life in the game of Go. *Information Sciences*, Vol. 10, No. 2, pp. 17–29. ISBN 0–387–96609–9, ISSN 0020–0255. Reprinted in D. N. L Levy, editor, *Computer Games*, Vol. II, pages 203–213, Springer-Verlag, New York, 1988.

Breuker, D. M. (1998). *Memory versus Search in Games.* Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands.

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (1996). Replacement schemes and two-level tables. *ICCA Journal*, Vol. 19, No. 3, pp. 175–180.

Campbell, M. (1985). The graph-history interaction: on ignoring position history. *Proceedings of the 1985 ACM Annual Conference on the Range of Computing: Mid-80's Perspective*, pp. 278–280, ACM Press, New York, NY.

Coulom, R. (2007). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Computers and Games (CG 2006)* (eds. H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers), Vol. 4630 of *Lecture Notes in Computer Science (LNCS)*, pp. 72–83, Springer-Verlag, Heidelberg, Germany.

Davies, J. (1994). 5×5 Go. *American Go Journal*, Vol. 28, No. 2, pp. 9–12.

Davies, J. (2001). The rules of Go. *The Go Player's Almanac 2001* (ed. R. Bozulich), pp. 191–194, Kiseido Publishing Company. ISBN 4–906574–40–8.

Donninger, C. (1993). Null move and deep search. *ICCA Journal*, Vol. 16, No. 3, pp. 137–143.

Drange, T. (2002). Mini-Go. http://www.mathpuzzle.com/go.html.

Drange, T. (2009). Personal communication.

Dyer, D. (1995). Searches, tree pruning and tree ordering in Go. *Proceedings of the Game Programming Workshop in Japan '95* (ed. H. Matsubara), pp. 207–216, Computer Shogi Association, Tokyo. http://www.andromeda.com/people/ddyer/go/search.html.

Gasser, R. (1996). Solving Nine Men's Morris. *Computational Intelligence*, Vol. 12, pp. 24–41.

Henderson, P., Arneson, B., and Hayward, R. B. (2009). Solving 8×8 Hex. *Proceedings of IJCAI 2009*, pp. 505–510.

Herik, H. J. van den, Uiterwijk, J. W. H. M., and Rijswijck, J. van (2002). Games solved: Now and in the future. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 277–311. ISSN 0004–3702.

Irving, G., Donkers, H. H. L. M., and Uiterwijk, J. W. H. M. (2000). Solving Kalah. *ICGA Journal*, Vol. 23, No. 3, pp. 139–148.

Kishimoto, A. and Müller, M. (2005). A Solution to the GHI Problem for Depth-First Proof-Number Search. *Information Sciences*, Vol. 175, No. 4, pp. 296–314.

Marsland, T. A. (1986). A review of game-tree pruning. *ICCA Journal*, Vol. 9, No. 1, pp. 3–19.

Müller, M. (1995). *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. Ph.D. thesis, ETH Zürich, Zürich, Switzerland.

Müller, M. (1997). Playing it safe: Recognizing secure territories in computer Go by using static rules and search. *Proceedings of the Game Programming Workshop in Japan '97* (ed. H. Matsubara), pp. 80–86, Computer Shogi Association, Tokyo.

Nash, J. (1952). Some Games and Machines for Playing Them. Technical Report D-1164, Rand Corp.

Nelson, H. L. (1985). Hash tables in Cray Blitz. *ICCA Journal*, Vol. 8, No. 1, pp. 3–13.

Romein, J. W. and Bal, H. E. (2003). Solving the Game of Awari using Parallel Retrograde Analysis. *IEEE Computer*, Vol. 36, No. 10, pp. 26–33.

Romstad, T. (2006). *An Introduction to Late Move Reductions.* http://www.glaurungchess.com/lmr.html.

Schadd, M. P. D., Winands, M. H. M., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Bergsma, M. H. J. (2008). Best Play in Fanorona Leads to Draw. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 369–387.

Schaeffer, J. (1983). The history heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16–19.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is Solved. *Science*, Vol. 317, No. 5844, pp. 1518–1522.

Schaeffer, J. and Plaat, A. (1996). New Advances in Alpha-Beta Searching. *Proceedings of the 1996 ACM 24th Annual Conference on Computer Science*, pp. 124–130. ACM Press, New York, NY, USA.

Sei, S. and Kawashima, T. (2000). A solution of Go on 4×4 board by game tree search program. *The 4th Game Informatics Group Meeting in IPS Japan*, pp. 69–76 (in Japanese). Translation available at http://homepage1.nifty.com/Ike/katsunari/paper/4x4e.txt.

Thorp, E. and Walden, W. E. (1972). A computer-assisted study of Go on M × N boards. *Information Sciences*, Vol. 4, No. 1, pp. 1–33.

Tromp, J. (2008). Solving Connect-4 on Medium Board Sizes. *ICGA Journal*, Vol. 31, No. 2, pp. 110–112.

Uiterwijk, J. W. H. M. and Herik, H. J. van den (2000). The advantage of the initiative. *Information Sciences*, Vol. 122, No. 1, pp. 43–58.

Wágner, J. and Virág, I. (2001). Solving Renju. *ICGA Journal*, Vol. 24, No. 1, pp. 30–34.

Werf, E. C. D. van der (2004). *AI techniques for the game of Go.* Ph.D. thesis, Maastricht University, Maastricht, The Netherlands. ISBN 90 5278 445 0.

Werf, E. C. D. van der, Herik, H. J. van den, and Uiterwijk, J. W. H. M. (2003). Solving Go on Small Boards. *ICGA Journal*, Vol. 26, No. 2, pp. 92–107.

Winands, M. H. M. (2008a). 6×6 LOA is solved. *ICGA Journal*, Vol. 31, No. 4, pp. 234–238.

Winands, M. H. M., Björnsson, Y., and Saito, J.-T. (2008b). Monte-Carlo Tree Search Solver. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 25–36, Springer-Verlag, Heidelberg, Germany.

Zobrist, A. L. (1970). A New Hashing Method with Application for Game Playing. Technical Report 88, Computer Science Department, University of Wisconsin, Madison, WI. Reprinted (1990) in *ICCA Journal*, Vol. 13, No. 2, pp. 69–73.