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

Solving Nine Men's Morris

RALPH GASSER

Abstract. We describe the combination of two search methods used to solve Nine Men's Morris. An improved retrograde analysis algorithm computed endgame databases comprising about 10^{10} states. An 18-ply alpha-beta search then used these databases to prove that the value of the initial position is a draw. Nine Men's Morris is the first non-trivial game to be solved that does not seem to benefit from knowledge-based methods.

1. Introduction

In recent years, a number of games have been solved using computers, including Qubic [Patashnik 1980], Connect-4 [Allen 1989; Allis 1988] and Go-Moku [Allis et al. 1993]. All these games were solved using knowledge-based methods. These methods are successful because all these games have a low decision complexity [Allis et al. 1991], that is, the right move is often easy to find. Not all games profit to the same extent. For instance, in checkers, chess and go a multitude of moves often seem fairly equal. Brute-force search is often the most viable means of playing or solving this type of game. This observation is supported by the fact that the best programs for both chess [Hsu 1990] and checkers [Schaeffer 1992] rely heavily on search.

Search methods are not only useful for playing games: they are ubiquitous in many aspects of problem solving. Some of these algorithms are among the best understood in computer science. However, not all search algorithms are equally well studied; in particular, exhaustive search in large state spaces is still in its infancy. Partly this is because the hardware has only recently progressed to a point where interesting problems are within reach. Due to the continual improvement in hardware and software, search methods must be constantly re-evaluated and adapted to new system designs [Stiller 1991; Lake 1992]. Games are ideal for gaining expertise in this area, because they are played in a restricted domain, where the problem difficulty can be controlled and the effectiveness of the approach measured.

We describe the combination of two brute-force search methods used to solve Nine Men's Morris. An improved retrograde analysis algorithm computed endgame databases comprising about 10^{10} states. An 18-ply alpha-beta search then used these databases to prove that the value of the initial position is a draw. Nine Men's Morris is the first non-trivial game to be solved that does not seem to benefit from knowledge-based methods.

2. Nine Men's Morris

Nine Men's Morris is one of the oldest games still played today [Bell 69]. Boards have been found on many historic buildings throughout the world. The oldest (about 1400 BC) was found carved into a roofing slate on a temple in Egypt. Others have been found as widely strewn as Ceylon, Troy and Ireland.

The game is played on a board with 24 points where stones may be placed. Initially the board is empty and each of the two players hold nine stones. The player with the white stones starts.

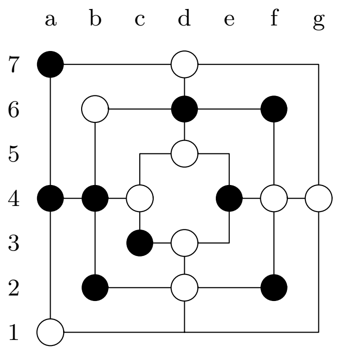

Figure 1. After the opening.

During the opening, players alternately place their stones on any vacant point (Figure 1).

After all stones have been placed, play proceeds to the midgame. Here a player may slide one of her stones to an adjacent vacant point. If at any time during the game a player succeeds in arranging three of her stones in a row—this is known as *closing a mill*—she may remove any opponent's stone that is not part of a mill. In Figure 2, if White closes a mill in the opening by playing to b6, she can now remove Black's stone on a1, but not the one on d2.

As soon as a player has only three stones left, the endgame commences. When it is her turn, the player with three stones may jump one of her stones to any vacant point on the board.

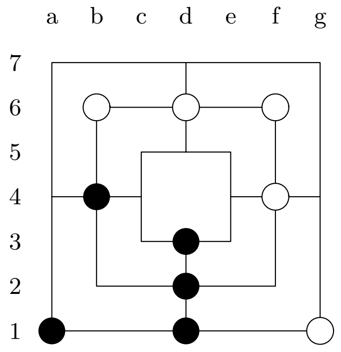

Figure 2. After b6, White removes a1 or b4.

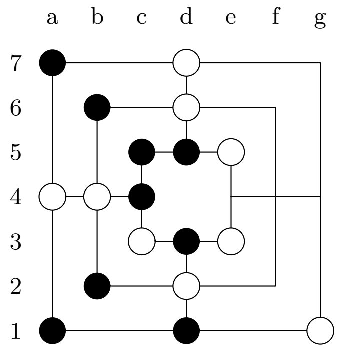

Figure 3. Black has no moves.

The game ends in the following ways:

- A player who has less than three stones loses.
- A player who cannot make a legal move loses (Figure 3).
- If a midgame or endgame position is repeated, the game is a draw.

Two points are subject to debate among Nine Men's Morris enthusiasts. The first hinges on the observation that in the opening it is possible to close two mills simultaneously. Should the player then be allowed to remove one or two opponent's stones? Our implementation only allows one stone to be removed. The second point concerns positions where the player to move has just closed a mill, but all the opponent's stones are also in mills. May she then remove a stone or not? In our implementation she may remove any stone. It seems unlikely that either of these rule variations affect the value of the game.

3. Solving Nine Men's Morris

Nine Men's Morris can be divided into two distinct phases. The opening, where stones are placed on the board and the mid- and endgame, where stones are moved. These two phases lead to a state space graph with specific characteristics. Most importantly, the opening phase induces an acyclic graph, whereas in the mid- and endgame phase move cycles can occur. Another difference is the search depth. The opening is clearly defined to be 18 plies deep for every path; by contrast, the time spent in the mid- and endgame depends on the chosen moves.

These different properties suggest the use of different search methods. We decided to use retrograde analysis to construct databases containing all mid- and endgame positions. This seems advantageous because retrograde analysis handles cycles more efficiently than forward search. In addition the opening search will require the values of many positions. Due to the interdependencies, this probably means computing the value for all or nearly all mid- and endgame positions, an ideal task for retrograde analysis.

Computing further databases for the opening is not reasonable, because their size would be even larger than for the mid- and endgame. In addition, because the value of only a single position (the empty board) is of interest in the opening phase, no intermediate position values must be stored. Alpha-beta search [Knuth and Moore 1975] is ideal for this type of problem.

4. Mid- and Endgame Databases

4.1. State space. To see if applying retrograde analysis is feasible, we must construct an appropriate state space. Each of the 24 board points can either be unoccupied or occupied by a black or white stone, so an upper bound for the state space size is 3^{24}, or approximately 2.8×10^{11}, states. This rather loose bound can be improved by taking the following game constraints into account:

- players have three to nine stones on the board,
- there are unreachable positions, and
- the board is symmetrical.

The first observation allows us to split the state space into 28 subspaces. Figure 4 shows these subspaces and their dependencies; thus the 7-5 positions (seven stones against five) can only be computed after the values for all 7-4 and 6-5 positions have been determined.

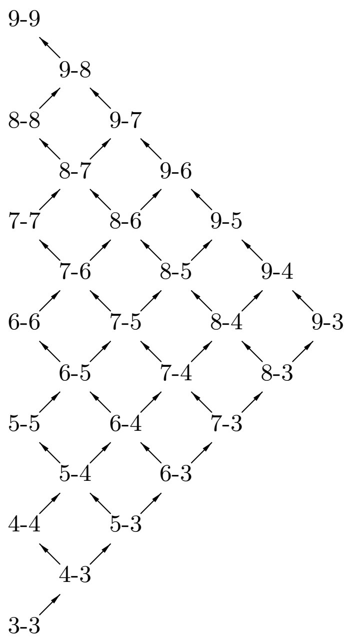

Figure 4. Database dependencies.

Not all states in these subspaces are reachable in the course of a legal game. For example, if one player has a closed mill, her opponent cannot have all nine stones on the board (Figure 5). Similar considerations can be made if two or three closed mills are on the board. The 9-9, 9-8, 9-7, 9-6, 9-5, 9-4, 9-3, 8-8, 8-7 and 8-6 subspaces contain such unreachable states.

The subspaces also contain many symmetrical positions. In general there are five symmetry axes (Figure 6). In the special case of the 3-3 subspace, there are additional symmetries, because all rings are interchangeable. Since one of these five axis is redundant—for example, $\pi_4 = \pi_1 \circ \pi_2 \circ \pi_3$, we can expect about a 16-fold reduction in the state space size from the symmetries.

Taking all three reductions into account leaves a total of 7,673,759,269 states in the mid- and endgame phase.

For such large state spaces, a memory-efficient storage method is needed. The standard method used in retrograde analysis is to construct a perfect hash function, which, given a state, returns a unique index. This makes it unnecessary to store the state description along with the state value, because the state de-

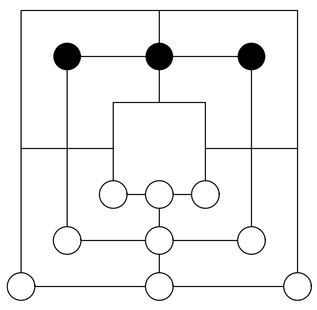

Figure 5. An unreachable state.

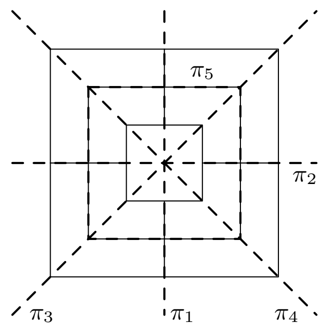

Figure 6. Symmetry axes.

scription is encoded in the index. Ideally the perfect hash function ranges from 0 to the number of states minus 1, and can be computed rapidly. Constructing such a hash function can be difficult. Therefore, as rapid computation is critical, it seems reasonable to loosen the requirements and allow hash functions with slightly larger ranges. This means that some unused entries are included in the file, but this will not impede the computation as long as the file size does not become too large. The hash function we decided to use maps the 7,673,759,269 states into a range of 9,074,932,579 indices.

4.2. Calculation. The first step in retrograde analysis is to initialize all easily recognizable won and lost positions. In Nine Men's Morris there are two types of terminal positions, both of which are losses for the player to move: positions where the player cannot move, and positions where the player has less than three stones. Therefore, if the player to move is blocked, we mark that state as a loss. We would also like to mark positions where the player to move has less than three stones as losses, but because we eliminated these positions from the state space, we mark positions where the player to move can close a mill and reduce the opponent to two stones as wins. We also mark positions where the player with three stones loses in two plies. Such positions are easy to recognize and belong to one of the following categories:

- the opponent has two open mills with no stones in common;
- the opponent has two open mills and the player to move cannot close a mill; or
- the player to move must remove a stone blocking an opponent's mill.

After the initialization has set these wins and losses, it sets the value of all remaining positions to a draw. An iterative process then determines the real values of these "drawn" states. As a first step, consider how wins and losses are propagated in two-player games. If a position is lost for the player to move, all the position's predecessors can be marked as wins for the opponent. Similarly, if a position is won for the player to move, all predecessors are potential losses for the opponent. They are only true losses if all of their successors are also won for the player (Figure 7).

It would be inefficient to check all successors to determine if a position is a loss or not. This can be avoided by using a Count field for each state, to store the number of successors that have not yet been proved a win for the opponent. Then, instead of checking all successors every time a position occurs, we simply decrement Count. When it reaches zero, all successors have been proved wins for the opponent, and the position can be marked a loss. The cost of this improvement is the additional memory used for the Count array.

Since we compute the depth of the wins and losses, one byte is used to store the value of each state, in a Val array. In practice the Count and Val array can use the same memory, since the two pieces of information are never needed

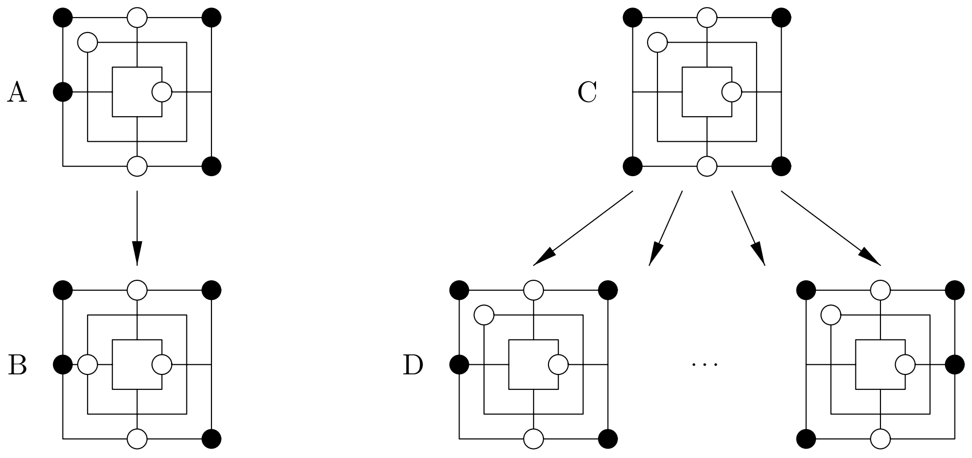

Figure 7. Determining predecessor values. Position A is won for White (to play) since *some* successor, say B, is lost for Black. Position C is lost for Black since *all* of it successors D are won for White.

simultaneously: the Count array is only needed while a position's value may still be drawn. In other words, the Count and Val values are stored as a "union"; see Table 1.

Table 2 shows the algorithm by which we determine the value of all states.

The first database was computed in 1989 on a Macintosh IIx, the last on a Macintosh Quadra 800 in 1993. In between, the algorithm was improved and ported to various machines. These include a Cray X-MP/28, a DEC VAX 9000-420, an IBM RS/6000, a 16-Transputer (T805) system, a 30-Processor MUSIC system (Motorola DSP 96000) and a DEC 3000 (Alpha). Only the DEC 3000 showed better performance than the Apple Macintosh. There are various reason for this surprising result. Many machines executed our code at low priority because of their multi-user environments. Also, many of the more powerful machines are optimized to deal with floating-point and vector operations, whereas their integer performance is not as highly tuned. The parallel machines additionally suffered from insufficient main memory and no direct disk access. This made the Macintosh front-end an I/O bottleneck.

Val entry	Count entry
0 = loss in 0	....
1 = win in 1	252 = "draw" (3 successors unknown)
2 = loss in 2	253 = "draw" (2 successors unknown)
3 = win in 3	254 = "draw" (1 successor unknown)
......	255 = "draw" (all successors unknown)

Table 1. All known information for a position fits in one byte.

```
Backupable ← [ ];                              {initialization}
for all states do
   Val[state] ← Initialize(state);
   Count[state] ← 0;
   if Val[state] ≠draw then
      Put(state,Backupable);
   end; {if}
end; {for}
while not Empty(Backupable) do                 {iteration}
   state ← GetState(Backupable);
   for all Predecessors(state) do
      if Val[pred] = draw then
         if Val[state] = loss then
            Val[pred] ← win;
            Put(pred, Backupable);
         else   {Val(state) = win}
            if Count[pred] = 0 then
               for all Successors(pred) do     {count successors}
                  Count[pred] ← Count[pred]+1;
               end; {for}
            end; {if}
            Count[pred] ← Count[pred]−1;
            if Count[pred] = 0 then
               Val[pred] ← loss;
               Put(pred,Backupable);
            end; {if}
         end; {if}
      end; {if}
   end; {for}
end; {while}
```

Table 2. This algorithm shows how the initialization and iteration determine the value of all states. For simplicity, we don't show the merging of the Count and Val arrays, nor the computation of the depth of the wins and losses.

4.3. Verification. Computations of this size almost inevitably contain some errors. The most common cause are software bugs in the retrograde analysis code, the system software, or the compiler. Hardware problems, such as disk errors or memory glitches, occur as well. For these reasons, the databases were verified on a cluster of 30 Sun Sparcstations [Balz 1994]. This took approximately three months. The verification found half a dozen hardware errors. Additionally, an error was found in the Nine Men's Morris code, which allowed some positions to be classified as draws instead of losses. These errors induced further inconsistencies, so that thousands of positions had to be corrected.

The verification only checked that the wins, losses and draws were consistent. The depth information was not checked, because the additional memory needed to store the databases increases the disk I/O and slows the verification dramatically. Presently, we are running a depth verification algorithm on a new parallel machine which may enable the complete verification. Even if depth verification finds no additional errors, how can we ascertain that none exist? A basic idea behind the verification is to use a different algorithm (forward search) and independent code to verify the data. But there are still some procedures common to both algorithms, such as the file indexing function or the initial state values. Ideally, our results should be independently verified.

5. Opening Search

Now that all mid- and endgame position values are computed and stored in databases, the value of the initial position can be found with a simple 18-ply alpha-beta search. In principle, the 18-ply opening can lead to any of the databases except 8-3, 9-3 and 9-4. This comprises approximately 9 GBytes of data. Since our machine only had 72 MBytes, accessing database values becomes an I/O bottleneck. The following methods were applied to alleviate this problem:

- reduce the size of the databases,
- reduce the number of used databases, and
- reduce the number of disk accesses.

Because no cycles occur in the opening, it is not necessary to compute the depth of the win or loss. Therefore a first idea is to reduce the size of the databases by packing five positions into one byte ($3^5 = 243$). However, in order to further minimize storage requirements, we decided to pack eight states into a byte. Although we then have only one bit per state, this is sufficient to compute the game-theoretic value if we do two alpha-beta searches. In the first search we determine if the initial position is won or not. If it is not won, we do a second search (with different databases) to determine if the initial position is lost or not.

While this reduces the size of the databases by a factor of eight, it still leaves files totaling about 1 GByte. To Nine Men's Morris players, it is clear that most

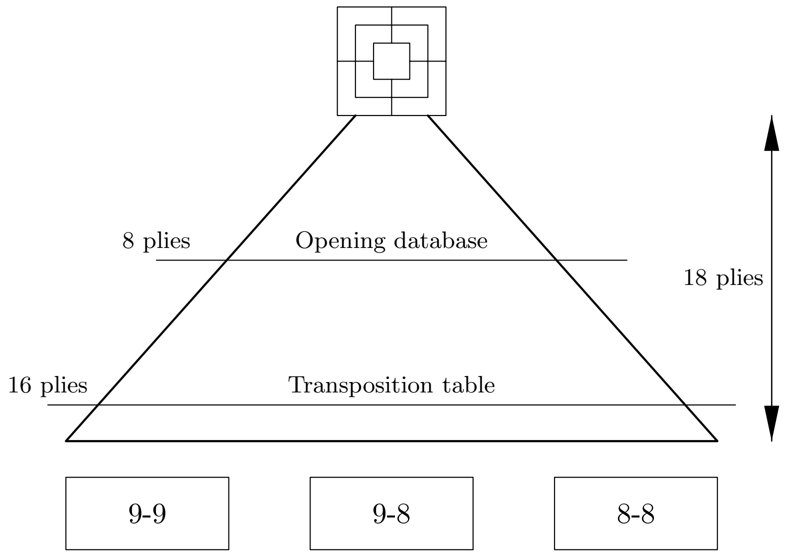

Figure 8. The opening search.

reasonable games will result in at most one or two mills being closed during the opening. For this reason we decided to try to prove the draw using only the 9-9, 9-8 and 8-8 databases, a total of 115 MBytes. It is then no longer possible to compute the correct value for every position. But in a position with White to move, we can compute an upper bound on the correct value by assuming all positions not in the databases are won for white and a lower bound if we assume all such positions are won for black. To prove that the game-theoretic value is a draw, we must show that White has both an upper and lower bound of a draw.

We inserted a transposition table at the 16-ply level to further reduce the number of database accesses. Normally, a transposition table includes positions from any ply, possibly even giving priority to those higher up in the tree. We chose to include only the 16-ply positions because it is far more important to reduce disk I/O than the total number of nodes in the tree. See Figure 8.

Finally, all positions that were visited by the alpha-beta searches at the 8-ply level were stored in an intermediate database. This allows games to be played in real time, because for the first 8 plies only a search to the intermediate database must be performed. Here is the number of 8-ply positions alpha-beta visited in each of the two searches:

	proved	not proved
White at least draws	15,501	12
White at most draws	4,364	29

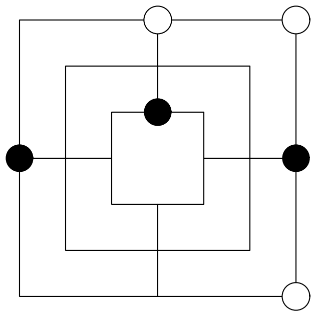
Figure 9. Black to move.
Loss in 26 plies.

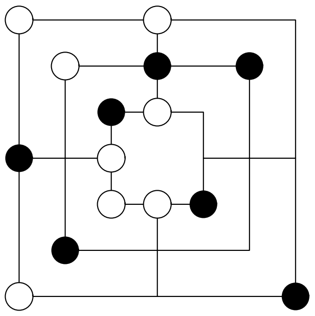
Figure 10. White to move and win.
Mill closure in 187 plies.

We differentiate between positions that could be proved to be at least (or at most) a draw and those that could not. This does not mean the positions are lost (or won); only that using three databases was insufficient to show otherwise.

Of the approximately 3.5 million positions at the 8-ply level, the table shows that only 19,906 were evaluated. The time for a node evaluation ranged from 2 seconds to 15 minutes. The total run-time was about three weeks on a Macintosh Quadra 800. Examining the table, one might be inclined to assume that most positions are drawn. This must not be the case; perhaps the move-ordering heuristic was simply successful in selecting drawish moves.

6. Results

6.1. Database Positions. Sifting through the final databases in search of "interesting" positions is a daunting task. Since we have no real intuition about what makes a position interesting, we decided to gather results in a more statistical fashion. The next two figures show examples of positions with long winning sequences. The 3-3 position in Figure 9 is an example of the longest sequence in that database. The 8-7 position in Figure 10 shows a won position with the longest move sequence until a mill is closed.

Figure 11 shows the percentage of positions that are won for the player to move. They are grouped according to the number of stones on the board, and normalized accordingly. The figure shows that the win probability strongly correlates with the stone difference: more stones are better. There seems to be a cut-off at about seven stones, so that having less makes winning difficult. The statistics also show that the 3-3 database seems to be special. This is probably because both players are allowed to jump, essentially making this a different game. One must keep in mind that these statistics can be misleading. For instance, in the 4-3 database, if the player with four stones is to move, it seems she has a small chance of winning. Upon closer examination of the database, we see that all wins are trivial positions where the player can close a mill immediately.

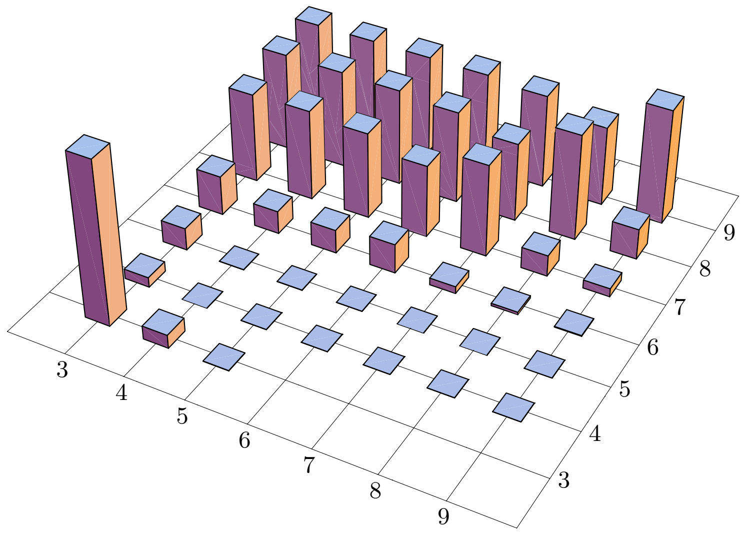

Figure 11. Win percentages for the player to move, as a function of the initial number of stones. The highest percentage, for 3-3 games, is about 83%. The absence of a column (as for 3-9 games) indicates a zero percentage.

These positions will not occur in a real game, so actually the player with three stones has a small advantage. Similar reservations apply to all these databases. It is not clear how the realistic positions can be filtered from the rest.

6.2. Opening Positions. It is now also possible to examine some common opening positions. Between equally matched players, games tend to be very drawish during the opening, that is, most possible moves result in a draw, as for the example in Figure 12. However, occasional blunders do occur. Figure 13 shows a position that occurred in a match our Nine Men's Morris program played against Mike Sunley (British Champion) at the Second Computer Olympiad in London, 1990. The program, which was then still based on heuristics, won four games and drew two.

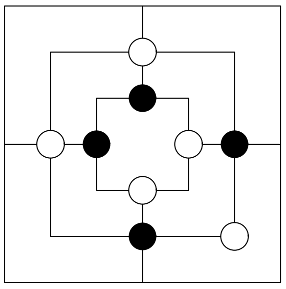

Figure 12. Opening position with black to move. Any move leads to a drawn position.

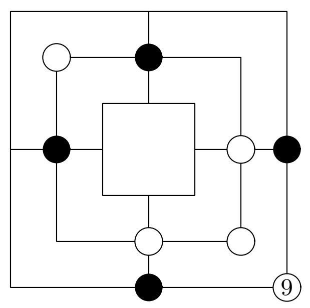

Figure 13. White 9 loses.

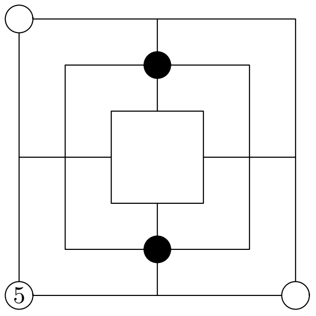

Figure 14. White 5 loses.

Since the initial position is a draw, a natural question to ask is how soon a mistake can be made. The position in Figure 14 shows one of the earliest losing moves we have found. This positions refutes a common misconception, namely that closing mills is a desirable opening strategy. White follows this strategy with her first two moves, while Black ignores the potential mill threats. Subsequently, White's third move loses even though she will now easily be able to close two mills during the opening.

7. Conclusion

Nine Men's Morris is a draw. This result was achieved using a combination of alpha-beta search and endgame databases. Our new efficient retrograde analysis algorithm allowed the 10^{10} database states to be solved on a desktop computer. This makes Nine Men's Morris the first classic board game to be solved that does not profit from knowledge-based methods.

The expertise gained in the management and use of large, pre-computed databases can also be applied to other problems. Many games and puzzles profit from combining databases with search, for instance Awari, Checkers, the 15-Puzzle or Rubik's Cube. Since games and puzzles are a restricted problem domain, future work will examine how more general optimization problems can be made amenable to this approach.

References

[Allen 1989] J. D. Allen, "A note on the computer solution of connect-four", pp. 134–135 in *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad* (edited by D. N. L. Levy and D. F. Beal), Ellis Horwood, Chichester, England, 1989.

[Allis 1988] L. V. Allis, "A knowledge-based approach to connect-four. The game is solved: White wins", M.Sc. thesis, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1988.

[Allis et al. 1991] L. V. Allis, H. J. van den Herik, and I. S. Herschberg, "Which games will survive", pp. 232–243 in *Heuristic Programming in Artificial Intelligence: The Second Computer Olympiad* (edited by D. N. L. Levy and D. F. Beal), Ellis Horwood, Chichester, England, 1991.

[Allis et al. 1993] L. V. Allis, H. J. van den Herik, and M. P. H. Huntjens, "Go-Moku solved by new search techniques", pp. 1–9 in *Proc. AAAI Fall Symposium on Games: Planning and Learning*, AAAI Press Tech. Report FS93-02, Menlo Park, CA.

[Balz 1994] G. Balz, "Verification of state databases", Diploma thesis, ETH Zürich, 1994.

[Bell 1969] R. C. Bell, *Board and Table Games from Many Civilisations*, Oxford University Press, Oxford, 1969.

[Hsu 1990] F. H. Hsu, "Large-scale parallelization of alpha-beta search: an algorithmic study with computer chess", Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, 1990.

[Knuth and Moore 1975] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning", *Artificial Intelligence* **6 1975**, 293–326.

[Lake et al. 1992] R. Lake, P. Lu, and J. Schaeffer, "Using retrograde analysis to solve large combinatorial search spaces", pp. 181–188 in *The 1992 MPCI Yearly Report: Harnessing the Killer Micros* (edited by E. D. Brooks et al.), Lawrence Livermore National Laboratory (UCRL-ID-107022-92), 1992.

[Patashnik 1980] O. Patashnik, "Qubic: $4 \times 4 \times 4$ tic-tac-toe", *Math. Mag.* **53** (1980), 202–216.

[Schaeffer et al. 1992] J. Schaeffer et al., "A World Championship caliber checkers program", *Artificial Intelligence* **53** (1992), 273–290.

[Stiller 1991] L. Stiller, "Group graphs and computational symmetry on massively parallel architecture", J. Supercomputing **5** (1991), 99–117.

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