Games of No Chance 3
MSRI Publications
Volume **56**, 2009

Tigers and Goats is a draw

LIM YEW JIN AND JURG NIEVERGELT

ABSTRACT. Bagha Chal, or "Moving Tiger", is an ancient Nepali board game also known as *Tigers and Goats*. We briefly describe the game, some of its characteristics, and the results obtained from an earlier computer analysis. As in some other games such as Merrill's, play starts with a placement phase where 20 pieces are dropped on the board, followed by a sliding phase during which pieces move and may be captured. The endgame sliding phase had been analyzed exhaustively using retrograde analysis, yielding a database consisting of 88,260,972 positions, which are inequivalent under symmetry. The placement phase involves a search of 39 plies whose game tree complexity is estimated to be of the order 10^{41}. This search has now been completed with the help of various optimization techniques. The two main ones are: confronting a heuristic player with an optimal opponent, thus cutting the search depth in half; and constructing a database of positions halfway down the search tree whose game-theoretic value is determined exhaustively. The result of this search is that *Tigers and Goats* is a draw if played optimally.

1. Introduction

Bagha Chal, or "Moving Tiger", is an ancient Nepali board game, which has recently attracted attention among game fans under the name *Tigers and Goats*. This game between two opponents, whom we call "Tiger" and "Goat", is similar in concept to a number of other asymmetric games played around the world — asymmetric in the sense that the opponents fight with weapons of different characteristics, a feature whose entertainment value has been known since the days of Roman gladiator combat.

On the small, crowded board of 5 x 5 grid points shown in Figure 1, four tigers face up to 20 goats. A goat that strays away from the safety of the herd and ventures next to a tiger gets eaten, and the goats lose if too many of them get swallowed up. A tiger that gets trapped by a herd of goats is immobilized, and the tigers lose if none of them can move. Various games share the characteristic that a multitude of weak pieces tries to corner a few stronger pieces, such as “Fox and Geese” in various versions, as described in “Winning ways” [BCG 2001] and other sources.

The rules of *Tigers and Goats* are simple. The game starts with the four tigers placed on the four corner spots (grid points), followed by alternating moves with Goat to play first. In a *placement phase*, which lasts 39 plies, Goat drops his 20 goats, one on each move, on any empty spot. Tiger moves one of his tigers according to either of the following two rules:

- A tiger can slide from his current spot to any empty spot that is adjacent and connected by a line.
- A tiger may jump in a straight line over any single adjacent goat, thereby killing the goat (removing it from the board), provided the landing spot beyond the goat is empty.

If Tiger has no legal move, he loses the game; if a certain number of goats have been killed (typically five), Goat loses.

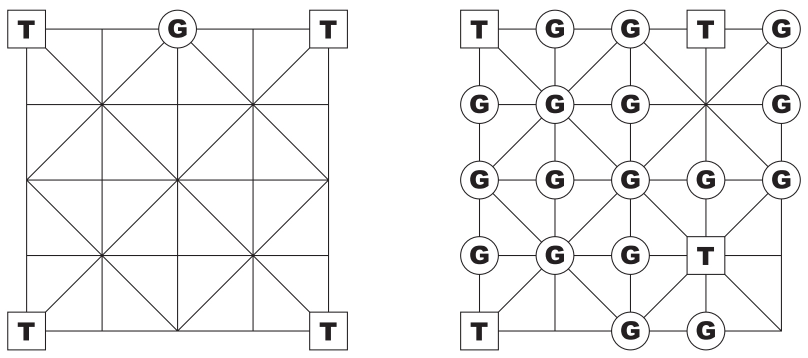

Figure 1. Left: The position after the only (modulo symmetry) first Goat move that avoids an early capture of a goat. At Right: Tiger to move can capture a goat, but thereafter Goat suffocates the tigers with a forcing sequence of 5 plies (challenge: find it).

These rules are illustrated in Figure 2, which also show that Goat loses a goat within 10 plies unless his first move is on the center spot of a border.

The 39-ply placement phase is followed by the *sliding phase* that can last forever. Whereas the legal Tiger moves remain the same, the Goat rule changes: on his turn to play, Goat must slide any of his surviving goats to an adjacent empty spot connected by a line. If there are 17 or fewer goats on the board, 4 tigers cannot block all of them and such a move always exists. In some exceptional cases (which arise only if Goat cooperates with Tiger) with 18 or more goats, the 4 tigers can surround and block off a corner and prevent any goat moves. Since Goat has no legal moves, he loses the game.

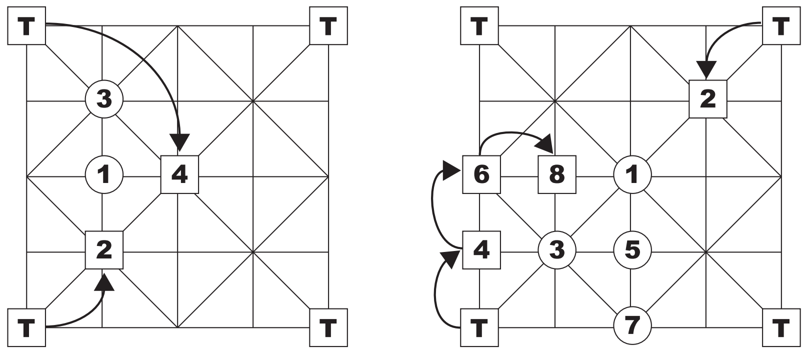

Figure 2. Goat has 5 distinct first moves (ignoring symmetric variants). All but the first move shown in Figure 1 lead to the capture of a goat within at most 10 plies, as these two forcing sequences show. At right, Tiger's last move 8 sets up a double attack against the two goats labeled 1 and 3.

Although various web pages that describe *Tigers and Goats* offer advice on how to play the game, we have found no expert know-how about strategy and tactics. Plausible rules of thumb about play include the following. First, it is obvious that the goats have to hug the border during the placement phase — any goat that strays into the center will either get eaten or cause the demise of some other goat. Goat's strategy sounds simple: first populate the borders, and when at full strength, try to advance in unbroken formation, in the hope of suffocating the tigers. Unfortunately, this recipe is simpler to state than to execute. In contrast, we have found no active Tiger strategy. It appears that the tigers cannot do much better than to wait, "doing nothing" (just moving back and forth), until near the end of the placement phase. Their goal is to stay far apart from each other, for two reasons: to probe the full length of the goats' front line for gaps, and to make it hard for the goats to immobilize all four tigers at the same time. Tiger's big chance comes during the sliding phase, when the compulsion to move causes some goat to step forward and offers Tiger a forcing sequence that leads to capture. Thus, it seems that Tiger's play is all tactics, illustrating chess Grandmaster Tartakover's famous pronouncement: "Tactics is what you do when there is something to do. Strategy is what you do when there is nothing to do".

2. Results of a previous investigation

Our earlier investigation with the goal of solving *Tigers and Goats* had given us partial results and a good understanding of the nature of this game, but we fell short of achieving an exhaustive analysis in the sense of determining the outcome: win, loss or draw, under optimal play. Here we summarize the main insights reported in [Lim 2004].

Size and structure of the state space. The first objective when attacking any search problem is to learn as much as possible about the size and structure of the state space in which the search takes place. For *Tigers and Goats* it is convenient to partition this space into 6 subspaces:

S_0: all the positions that can occur during the placement phase, including 4 tigers and 1 to 20 goats.
S_k: for $k = 1 \dots 5$, all the positions that can occur during the sliding phase, with 4 tigers, $21 - k$ goats, and k empty spots on the board.

Notice that any position in any S_1 to S_5 visually looks exactly like some position in S_0, yet the two are different positions: in S_0, the legal Goat moves are to drop a goat onto an empty spot, whereas in S_1 to S_5, the legal moves are to slide one of the goats already on the board. For each subspace S_1 to S_5, a position is determined by the placement of pieces on the board, which we call the board image, and by the player whose turn it is to move. Thus, the number of positions in S_1 to S_5 is twice the number of distinct board images.

For S_0, however, counting positions is more difficult, since the same board image can arise from several different positions, depending on how many goats have been captured. As an example consider an arbitrary board image in S_5, hence with 16 goats and 5 empty spots. This same board image could have arisen, as an element of S_0, from ten different positions, in which 0, 1, 2, 3, or 4 goats have been captured, and in each case, it is either Tiger's or Goat's turn to move. Although for board images in S_1 through S_4 the multiplier is less than 10, these small subspaces do not diminish the average multiplier by much. Thus, we estimate that the number of positions in S_0 is close to 10 times the number of board images in S_0, which amounts to about 33 billion.

Since the game board has all the symmetries of a square that can be rotated and flipped, many board positions have symmetric "siblings" that behave identically for all game purposes. Thus, all the spaces S_0 to S_5 can be reduced in size by roughly a factor of 8, so as to contain only positions that are pairwise inequivalent. Using Polya's counting theory [Polya 1937] we computed the exact size of the symmetry-reduced state spaces S_1 to S_5, and of the board images of S_0, as shown in Table 1.

S_0 is very much larger than all of S_1 to S_5 together, and has a more complex structure. Due to captures during the placement phase, play in S_0 can proceed back and forth between more or fewer goats on the board, whereas play in the sliding phase proceeds monotonically from S_k to S_{k+1}. These two facts suggest that the subspaces are analyzed differently: S_1 to S_5 are analyzed exhaustively using retrograde analysis, whereas S_0 is probed selectively using forward search [Gasser 1996].

	# of board images	# of positions
S_0	3,316,529,500	∼33,000,000,000
S_1	33,481	66,962
S_2	333,175	666,350
S_3	2,105,695	4,211,390
S_4	9,469,965	18,939,930
S_5	32,188,170	64,376,340

Table 1. Number of distinct board images and positions for corresponding subspaces

Database and statistics for the sliding phase. Using retrograde analysis [Wu 2002] we determined the game-theoretic value of each of the 88,260,972 positions in the spaces S_1 to S_5, i.e., during the sliding phase. A Tiger win is defined as the capture of 5 goats, a Tiger loss as the inability to move, and a draw (by repetition) is defined as a position where no opponent can force a win, and each can avoid a loss. Table 2 shows the distribution of won, drawn and lost positions.

		Number of goats captured				
		4	3	2	1	0
Goat to move	Wins	913,153 (2.8%)	1,315,111 (13.9%)	882,523 (41.9%)	252,381 (75.8%)	30,609 (91.4%)
	Draws	8,045,787 (25.0%)	6,226,358 (65.7%)	1,199,231 (57.0%)	80,706 (24.2%)	2,812 (8.4%)
	Losses	23,229,230 (72.2%)	1,928,496 (20.4%)	23,941 (1.1%)	88 (0.03%)	60 (0.2%)
	Total	32,188,170	9,469,965	2,105,695	333,175	33,481
Tiger to move	Wins	30,469,634 (94.7%)	6,260,219 (66.1%)	465,721 (22.1%)	6,452 (1.9%)	146 (0.4%)
	Draws	1,569,409 (4.9%)	2,918,104 (30.8%)	1,353,969 (64.3%)	197,537 (59.3%)	9,468 (28.3%)
	Losses	149,127 (0.5%)	291,642 (3.1%)	286,005 (13.6%)	129,186 (38.8%)	23,867 (71.3%)

Table 2. Endgame database statistics. Percentages are relative to totals for a given player to move.

Goats captured	1	2	3	4	5
Complexity	1.28×10^{24}	4.23×10^{36}	8.92×10^{38}	3.09×10^{40}	4.88×10^{41}

Table 3. Estimated tree complexity for various winning criteria.

Game tree complexity. The search space S_0, with approximately 33 billion positions, is too large for a static data structure that stores each position exactly once. Hence it is generated on the fly, with portions of it stored in hash tables. As a consequence, the same position may be generated and analyzed repeatedly. A worst case measure of the work thus generated is called game tree complexity. The size of the full search tree can be estimated by a Monte Carlo technique as described by [Knuth 1975]. For each of a number of random paths from the root to a leaf, we evaluate the quantity $F = 1 + f_1 + f_1 \times f_2 + f_1 \times f_2 \times f_3 + \cdots$, where f_j is the fan out, or the number of children, of the node at level j encountered along this path. The average of these values F, taken over the random paths sampled, is the expected number of nodes in the full search tree. Table 3 lists the estimated game tree complexity (after the removal of symmetric positions) of five different "games", where the game ends by capturing 1 to 5 goats during the placement phase. These estimates are based on 100,000 path samples.

Cutting search trees in half. A 39-ply search with a branching factor that often exceeds a dozen legal moves is a big challenge. Therefore, the key to successful forward searches through the state space S_0 of the placement phase is to replace a 39-ply search with a number of carefully designed searches that are effectively only 20 plies deep. This is achieved by 1) formulating hypotheses of the type "player X can achieve result Y", 2) programming a competent and efficient heuristic player X that generates only one or a few candidate moves in each position, and 3) confronting the selective player X with his exhaustive opponent who tries all his legal moves. If this search that alternates selective and exhaustive move generation succeeds, the hypothesis Y is proven. If not, one may try to develop a stronger heuristic player X, or weaken the hypothesis, e.g. from "X wins" to "X can get at least a draw". Using such searches designed to verify a specific hypothesis we were able to prove several results including the following:

(i) Tiger can force the capture of a single goat within 30 plies, but no sooner.
(ii) Tiger can force the capture of two goats within 40 plies, i.e., by the end of the placement phase, but not earlier.
(iii) After the most plausible first two moves (the first by Goat, the second by Tiger) Goat has a drawing strategy.

Heuristic attackers and defenders. In order to make these searches feasible we had to develop strong heuristic players. Given our lack of access to human expertise, we developed player programs that learn from experience by being pitted against each other — a topic described in [Lim 2005]. For example, the proof that Tiger can kill a certain number of goats requires a strong Tiger that tries to overcome an exhaustive Goat. Conversely, the proof that Goat has a drawing strategy after the most plausible opening requires a strong heuristic Goat that defies an exhaustive Tiger.

Goat has at least a draw. After Goat's most reasonable first move, Tiger has 6 symmetrically distinct replies at ply 2. Using the same techniques and software as described above, further computer runs that stretched over a couple of months proved that Goat has a successful defense against all of them. Having shown that Goat can ensure at least a draw, the next question is "does Goat have a winning strategy?".

Insights into the nature of the game. We were unable to discover easily formulated advice to players beyond plausible rules-of-thumb such as "goats cautiously hug the border, tigers patiently wait to spring a surprise attack". On the other hand, our database explains the seemingly arbitrary number "five" in the usual winning criterion "Tiger wins when 5 goats have been killed". This magic number "5" must have been observed as the best way to balance the chances. We know that Tiger can kill some goats, so Tiger's challenge must be more ambitious than "kill any one goat". On the other hand, we see from Table 2 that there is a significant jump in number of lost positions for Goat from three goats captured to four goats captured. It is therefore fairly safe to conjecture that once half a dozen goats are gone, they are all gone — Goat lacks the critical mass to put up resistance. But as long as there are at least 16 goats on the board (at most 4 goats have been captured), the herd is still large enough to have a chance at trapping the tigers.

Table 2 also shows that unless Tiger succeeds in capturing at least two goats during the placement phase, he has practically no chance of winning. If he enters the sliding phase facing 19 goats, less than 2% of all positions are won for Tiger, regardless of whether it is his turn to move or not. The fact that Tiger can indeed force the capture of 2 goats within 40 plies, that is, by the end of the placement phase (see page 168), is another example of how well-balanced the opponents' chances are.

3. Proving Tiger's draw

The previous investigation, with the result that Goat has at least a draw, had brought us tantalizingly close to determining the game-theoretic value of *Tigers* *and Goats*. Computing the endgame database had been relatively straightforward, but the 39-ply forward search had not yielded to the judicious application of established techniques. Experience had shown that by approximating a 39-ply search by various 19-ply and 20-ply searches (see "Cutting search trees in half", page 168), we were able to answer a variety of questions. It appeared plausible that by formulating sufficiently many well-chosen hypotheses this approach would eventually yield a complete analysis of the game. We conjectured that Tiger also has a drawing strategy, and set out to try to prove this using the same techniques that had yielded Goat's drawing strategy.

The asymmetric role of the two opponents, however, made itself felt at this point: the searches pitting a heuristic Tiger player against an exhaustive Goat progressed noticeably more slowly than those involving a heuristic Goat versus an exhaustive Tiger. In retrospect we interpret this different behavior as due to the phenomenon "Tiger's play is all tactics". Positional considerations — keep the goats huddled together — make it easy to generate one or a few "probably safe" Goat's moves, even without any look-ahead at the immediate consequences. For Tiger, on the other hand, neither we nor apparently the neural network that trained the player succeeded in recognizing "good moves" without a local search. An attempt to make Tiger a stronger hunter (by considering the top 3 moves suggested by the neural network followed by a few plies of full-width search) is inconsistent with the approach of "cutting the tree in half" and made the search unacceptably slow.

Thus, a new approach had to be devised. The experience that 20-ply forward searches proved feasible suggests a more direct approach: compute a database of positions of known value halfway down the search tree. Specifically, we define *halfway position* as one arising after 19 plies, i.e., after the placement of 10 goats, with Tiger to move next. The value of any such position can be computed with a search that ends in the endgame database after at most 20 plies. If sufficiently many such "halfway positions" are known and stored, searches from the root of the tree (the starting position of the game) will run into them and terminate the search after at most 19 plies.

The problem with this approach is that the number of halfway positions is large, even after symmetric variants have been eliminated. Because of captures not all 10 goats placed may still be on the board, hence a halfway position has anywhere between 6 and 10 goats, and correspondingly, 15 to 11 empty spots. Using the terminology of Section 2, the set of halfway positions is (perhaps a subset of) the union of S_{11}, S_{12}, S_{13}, S_{14} and S_{15}, where S_k is the set of all symmetrically inequivalent positions containing 4 tigers, $21-k$ goats, and k empty spaces. S_{11}, with about equally as many goats as empty spots, is particularly large. On the assumption that in any subspace S_k the number of symmetrically inequivalent positions is close to 1/8 of the total, S_{11} contains about 550 million inequivalent positions. The union of S_{11} through S_{15} contains about 1.6×10^9 positions. This number is about 25 times larger than the largest endgame database we had computed before, namely S_5.

The approach to overcome the problem of constructing a large halfway database exploits two ideas. First, the database of halfway positions of known value need not necessarily include all halfway positions. In order to prove that Tiger has a drawing strategy, the database need only include a sufficient number of positions known to be drawn or a win for Tiger so that any forward search is trapped by the filter of these positions. Second, the database of halfway positions is built on the fly: whenever a halfway position is encountered whose value is unknown, this position is entered into the database and a full-width search continues until its value has been computed.

Although there was no a priori certainty that this approach would terminate within a reasonable time, trial and error and repeated program optimization over a period of five months led to success. Table 4 contains the statistics of the halfway database actually constructed. For each of S_{15} through S_{11}, it shows the number of positions whose value was actually computed, broken down into the two categories relevant from Tiger's point of view, win-or-draw vs. loss.

# Captured	# Win or Draw	# Loss	Total	Estimated state space size
4	17,902,335	0	17,902,335	85,804,950
3	33,152,214	0	33,152,214	183,867,750
2	64,336,692	17,944	64,354,636	321,768,563
1	84,832,697	329,183	85,161,880	464,776,813
0	15,857,243	91,676	15,948,919	557,732,175
Total	216,081,181	438,803	216,519,984	1,613,950,251

Table 4. Halfway database statistics: the number of positions computed and their value from Tiger's point of view: win-or-draw vs. loss

Although the construction of the halfway database is intertwined with the forward searches — a position is added and evaluated only as needed — logically it is clearest to separate the two. We discuss details of the forward searches in the next section.

4. Implementation, optimization, verification

Our investigation of *Tigers and Goats* has been active, on and off, for the past three years. The resources used have varied form a Pentium 4 personal computer to a cluster of Linux PC workstations. Hundreds of computer runs were used to explore the state space, test and confirm hypotheses, and verify results. The longest continuous run lasted for five months as a background process on an Apple PowerMac G5 used mainly for web surfing.

The algorithmic search techniques used are standard, but three main challenges must be overcome in order to succeed with an extensive search problem such as *Tigers and Goats*. First, efficiency must be pushed to the limit by adapting general techniques to the specific problem at hand, such as the decision described above on how to combine different search techniques. Second, programs must be optimized for each of the computer systems used. Third, the results obtained must be verified to insure they are indeed correct. We address these three issues as follows.

Domain-specific optimizations. The two databases constructed, of endgame positions and halfway positions, limit all forward searches to at most 20 plies. Still, performing a large number of 20-ply searches in a tree with an average branching factor of 10 remains a challenge that calls for optimization wherever possible.

The most profitable source of optimizations is the high degree of symmetry of the game board. Whereas the construction of the two databases of endgame and halfway positions is designed to avoid symmetric variants, this same desirable goal proved not to be feasible during forward searches — it would have meant constructing a database consisting of all positions.

Instead, the goal is to avoid generating some, though not necessarily all, symmetrically equivalent positions when this can be done quickly, namely during move generation. Although the details are cumbersome to state, in particular for Tiger moves, the general idea is straightforward. Any position that arises during the search is analyzed to determine all active symmetries. Thereafter, among all the moves that generate symmetric outcomes, only that one is retained that generates the resulting position of lowest index. This analysis guarantees that all immediate successors to any given position are inequivalent. Because of transpositions, of course, symmetric variants will appear among successor positions further down in the tree. Table 5 shows the effect of this symmetry-avoiding move generation for the starting position. Although there is a considerable reduction in the number of positions generated, the relative savings diminish with an expanding horizon.

Ply	Naïve move generator	Symmetry-avoiding move generator	Number of distinct positions
1	21	5	5
2	252	36	33
3	5,052	695	354
4	68,204	9,245	2,709
5	1,304,788	173,356	18,906
6	18,592,000	2,441,126	93,812

Table 5. Number of positions created by different move generators.

System-specific optimization. Our previous result for *Tigers and Goats* used a cluster of eight Linux PC workstations with a simple synchronous distributed game-tree search algorithm. However, there are fundamental problems with synchronous algorithms, discussed in [Brockington 1997], that limit their efficiency. Furthermore, the cluster was becoming more popular and was constantly overloaded. We therefore decided against implementing a more sophisticated asynchronous game-tree search and instead relied on a sequential program running on a single dedicated processor.

We focused our attention on improving the sequential program to run on an Apple PowerMac G5 1.8 GHz machine running Mac OS-X. Firstly, the neural network code was optimized using the Single Instruction Multiple Data (SIMD) unit in the PowerPC architecture called AltiVec. AltiVec consists of highly parallel operations which allow simultaneous execution of up to 16 operations in a single clock cycle. This provided a modest improvement of about 15% to the efficiency of neural network evaluations of the board, but sped up the overall efficiency of the search much more as the neural network is used repeatedly within the search to evaluate and reorder the moves.

Next, we moved many of the computations off-line. For example, the moves for Tiger at each point on the board in every combination of surrounding pieces were precomputed into a table so that the program simply retrieved the table and appended it to the move list during search. Operations like the indexing of the board and symmetry transformation were also precomputed so that the program only needed to retrieve data from memory to get the result. Finally, we recompiled the software with G5-specific optimizations.

Verification. Two independent re-searches confirm different components of the result. They used separately coded programs written in C, and took 2 months to complete.

The first verification search used the database of halfway positions to confirm the result at the root, namely, "Tiger has a drawing strategy". Notice that this verification used only the positions marked as win-or-draw in the database.

The second verification search confirmed the halfway positions marked as win-or-draw by searching to the endgame database generated by the retrograde analysis described in [Lim 2004]. All other positions can be ignored, as they have no effect on the first search.

Another program was written in C to 'reprove' the results. This program had the benefit of a posteriori knowledge that the game is a draw, and this fact allowed us to concentrate on using aggressive forward pruning techniques to verify the result. The program used the same domain-specific optimizations such as symmetry reduction and the halfway databases.

The halfway database was optimized for size by storing the boolean evaluation of each position using a single bit. Depending on the type of search, this boolean evaluation could mean "Goat can at least draw" or "Tiger can at least draw". Due to this space optimization the halfway positions and endgame databases could be stored in memory, thereby avoiding disk accesses and speeding up the search by orders of magnitude.

As Tiger is able to force the capture of two goats only by the end of the placement phase, at ply 40, the search for "Goat can at least draw" used an aggressive forward pruning strategy of pruning positions which had two or more goats already captured. The halfway database was set at ply 23, when 12 goats have already been placed and it is Tiger's turn to move. The search confirmed that "Goat can at least draw" in approximately 7 hours while visiting 7,735,443,119 nodes.

The program was also able to confirm that "Tiger can at least draw". Due to the large game-tree complexity of this search, two intermediate databases were placed at ply 21 and ply 31. These databases contribute towards efficiency in two ways: first, they terminate some searches early, and second, they generate narrower search trees. The latter phenomenon is due to the fact that these databases are free of symmetrically equivalent positions. In exchange for a large memory footprint of approximately 2 GB, search performance was dramatically improved. The searched confirmed that "Tiger can at least draw" in approximately 48 hours while visiting 40,521,418,103 nodes.

5. Conclusion

The theory of computation has developed powerful techniques for estimating the asymptotic complexity of problem classes. By contrast, there is little or no theory to help in estimating the concrete complexity of computationally hard problem instances, such as determining the game-theoretic value of *Tigers and Goats*. Although the general techniques for attacking such problems have been well-known for decades, there are only rules of thumb to guide us in adapting them to the specific problem at hand in an attempt to optimize their efficiency [Nievergelt 2000].

The principal rule of thumb we have followed in our approach to solving *Tigers and Goats* is to precompute the solutions of as many subproblems as can be handled efficiently with the storage available, both in main memory (hash-tables) and disks (position data bases). If the net of these known subproblems is dense enough, it serves to truncate the depth of many forward searches, an effect that plays a decisive role since the computation time tends to grow exponentially with search depth. Beyond such rules of thumb, at the present state of knowledge about exhaustive search there is not much more we can do than persistent experimentation. Developing a technology that gives us quantitative estimates of the complexity of computationally hard problems remains a challenge.

Acknowledgment

Elwyn Berlekamp pointed out *Tigers and Goats* and got us interested in trying to solve this game — an exhaustive search problem whose solution stretched out over three years. We are grateful to Elwyn, Tony Tan, Thomas Lincke and H. J. van den Herik for helpful comments that improved this paper. Some of the present text is taken from our earlier paper [Lim 2004].

Note about references

We are not aware of any widely available publications on *Tigers and Goats*. Searching the web for *Tigers and Goats*, or Bagha Chal in various spellings, readily leads to a collection of web sites that describe the game and/or let you play against a computer program.

References

[BCG 2001] E. Berlekamp, J. H. Conway, R. K. Guy, *Winning Ways For Your Mathematical Plays*, A K Peters, 4 volumes, 2nd edition, 2001.

[Brockington 1997] M. G. Brockington, *Asynchronous parallel game-tree search*, Ph.D. Thesis, Department of Computing Science, University of Alberta, 1997.

[Gasser 1996] R. Gasser, *Solving Nine Men's Morris*, pp. 101–113 in Games of No Chance, edited by Richard Nowakowski, MSRI Publications 29, Cambridge University Press, New York, 1996.

[Knuth 1975] D. E. Knuth, *Estimating the efficiency of backtrack programs*, Math. Comp. 29, 1975, 121–136.

[Lim 2004] Y. J. Lim and J. Nievergelt, *Computing* Tigers and Goats, ICGA Journal 27:3, 131–141, Sep 2004.

[Lim 2005] Y. J. Lim, *Using biased two-population co-evolution to evolve heuristic game players for* Tigers and Goats, Unpublished manuscript.

[Nievergelt 2000] J. Nievergelt, *Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power*, pp. 18–35 in Sofsem 2000: Theory and Practice of Informatics, edited by V. Hlavac, K.G. Jeffery and J. Wiedermann, Lecture Notes in Computer Science 1963, Springer, Berlin, 2000.

[Polya 1937] G. Polya, *Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen*, Acta Mathematica 68, 1937, 145–253.

[Wu 2002] R. Wu and D. F. Beal, *A Memory efficient retrograde algorithm and its application to Chinese Chess endgames*, pp. 213–227 in More Games of No Chance, edited by Richard Nowakowski, MSRI Publications 42, Cambridge University Press, New York, 2002.

LIM YEW JIN
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE
SINGAPORE
limyewjin@gmail.com

JURG NIEVERGELT
INFORMATIK ETH
8092 ZURICH
SWITZERLAND
jn@inf.ethz.ch