Complexity and Retrograde Analysis of the Game Dou Shou Qi

Jan N. van Rijn Jonathan K. Vis

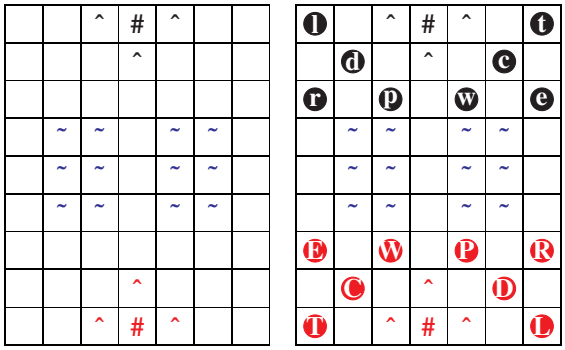
Leiden Institute of Advanced Computer Science, Universiteit Leiden, The Netherlands

Abstract

Dou Shou Qi is a game in which players control a number of pieces, aiming to move one of these onto a certain square. We will present a proof showing that this game is PSPACE-hard. Furthermore, we have implemented an analyzing engine and created an endgame tablebase containing all configurations with up to four pieces. These are the first steps towards theoretically solving the game. Finally, we report on some interesting patterns which we found by analyzing the endgame tablebase.

1 Introduction

Dou Shou Qi [7] (meaning: "Game of Fighting Animals") is a Chinese board game. In the Western world it is often called Jungle, The Jungle Game, Jungle Chess, or Animal Chess. Dou Shou Qi is a two player abstract strategy game and it contains some elements from Chess and Stratego as well as some other chess-like Chinese games (e.g., Banqi). Its origins are not entirely clear, but it seems that it evolved rather recently (around the 1900s) in China. Dou Shou Qi is played on a rectangular board consisting of 9×7 squares, see Figure 1. The columns are called *files* and are labelled a–g from left to right. The rows or *ranks* are numbered 1–9 from bottom to top (the board is viewed from the position of the white player).



(a) Empty game board (b) Initial position

Figure 1: A schematic Dou Shou Qi game board showing the empty game board and the initial configuration. For graphical reasons, white pieces are displayed in red.

There are several different kinds of squares. The *dens* (#) are located in the center of the first and the last rank ($d1$ and $d9$). Each den is surrounded by *traps* (^). There are also two rectangular (3×2 squares) bodies of *water* (~) sometimes called *rivers*. The remaining squares are ordinary land squares. Each player has eight different pieces representing different animals. Each animal has a certain *strength*, according to which

they can *capture* other (opponent's) pieces. Only pieces with the same or a higher strength may capture an opponent's piece. The only exception to this rule regards the weakest (rat) and the strongest (elephant) pieces. Just like the spy in Stratego, the weakest piece may capture the strongest. The strength of the pieces, from weak to strong, is: 1 `R`, `r` — Rat (sometimes called mouse); 2 `C`, `c` — Cat; 3 `W`, `w` — Wolf (sometimes called fox); 4 `D`, `d` — Dog; 5 `P`, `p` — Panther (sometimes called leopard); 6 `T`, `t` — Tiger; 7 `L`, `l` — Lion; 8 `E`, `e` — Elephant. The initial placement of the pieces is fixed, see Figure 1(b). The capital letters are used to denote the white pieces. Players alternate moves with white moving first. Each turn one piece must be moved. Each piece can move one square either horizontally or vertically. In principle a piece may not move into the water, and it is also forbidden to enter its own den ($d1$ for white, and $d9$ for black). The rat is the only piece that can swim, and is therefore able to enter the water. It may also capture in the water (the opponent's rat), however, it may not capture the elephant from the water. Lions and tigers are able to leap over water (either horizontally or vertically). They cannot jump over the water when a rat (own or opponent's) is on any of the intermediate water squares. When a piece is in an opponent's trap ($c9$, $d8$, $e9$ for white and $c1$, $d2$, $e1$ for black), its strength is effectively reduced to zero, meaning that any of the opponent's pieces may capture it regardless its strength. A piece in one of its own traps is unaffected. The objective of the game is to either place one of the pieces in the opponent's den or to eliminate all of the opponent's pieces. As in Chess, stalemate positions are declared a draw. A threefold repetition rule is imposed in some variants of this game. The existence of such a rule is irrelevant for our analyses.

The game Dou Shou Qi is not extensively studied in literature. In [1], the game is introduced and an attempt is made to characterize certain local properties of subproblems that occur when analyzing the game. These so-called loosely coupled subproblems can be analyzed separately in contrast to analyzing the problem as a whole resulting in a possible speed-up in the overall analysis. The authors also propose an evaluation (utility) function for Dou Shou Qi, which we will use in our research as well. We will also present an engine without taking the loosely coupled subproblems into account. A first complexity result has been obtained by the current author in [13]. Dou Shou Qi is proven PSPACE-hard by reduction from a game called Bounded 2CL, a graph game introduced by the authors of [3]. Our main contribution is a self-contained PSPACE-hardness proof by reduction from the well-known logic circuits. Furthermore, we will present an endgame tablebase containing all configurations up to four pieces.[1]

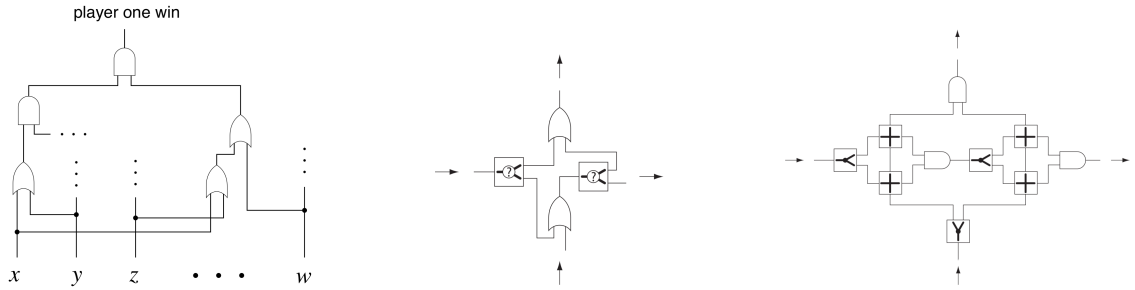
The remainder of the paper is organized as follows. In Section 2 we derive a theoretical complexity class for this game. We present the construction of an analysing engine and the endgame tablebases in Section 3 and Section 4, and the conclusions to the study in Section 5.

2 Complexity Analysis

We will show Dou Shou Qi to be PSPACE-hard, by reduction from G_{pos}(POS CNF) [8], which is defined as follows: Let A be a positive CNF formula, i.e., there are no negated variables. The two players take turns choosing a variable in A that has not yet been chosen. After all variables in A have been considered, variables chosen by the white player are set to true, and variables chosen by the black player are set to false. The white player wins if and only if A is true.

The author of [4] presents, among other complexity results, a framework for the reduction of games from G_{pos}(POS CNF). It provides a way to represent a positive CNF formula as a logic circuit, consisting of AND and OR gates. Furthermore, it introduces an additional logic gate, the *CHOICE gate*. The CHOICE gate consists of one input and two outputs. When the input is activated, one of the outputs can be activated, but not both. Given a positive CNF formula A, it will be represented as a logic circuit, as shown in Figure 2(a). We use standard digital logic symbols for the AND and OR gates. Each variable is represented exactly once on the circuit by a so-called *variable setter*. If a variable is selected by the white player, a signal is enabled to flow out from it; if the black player selects it, the signal is blocked. The variables are linked to each clause in which it is present. A binary clause is represented as an OR gate; clauses with a higher arity will be represented by a group of subsequent OR-gates. Finally, all clauses are linked together with logic AND gates. Players take turns setting a variable setter that has not yet been set by another player. The white player

[1] For the implementation of the engine and the retrograde analysis see: `http://www.liacs.nl/home/jvis/doushouqi`.



(a) Circuit corresponding to the CNF formula $(x \vee y) \wedge \ldots \wedge (x \vee z \vee w)$

(b) Half crossover

(c) Crossover

Figure 2: Reduction schematic introduced by [4].

decides for each activated CHOICE gate which output will be activated. He wins if a signal reaches the final output of the circuit. This is possible if and only if he can win in the formula game.

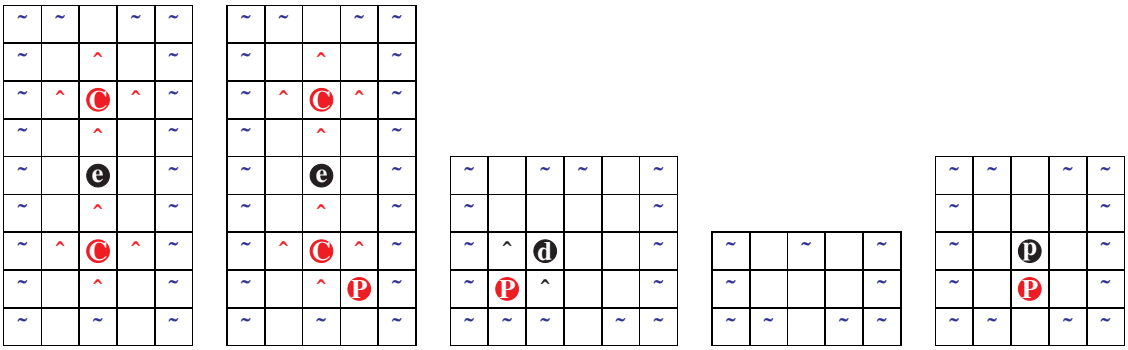
Logic circuits enable wires to cross each other, which is a property not easily preserved on a two dimensional grid, such as the Dou Shou Qi board. Using the CHOICE gate, the author of [4] constructs a so-called *half crossover*, as shown in Figure 2(b). CHOICE gates are displayed as a forking symbol with a question mark on it. The half crossover has two inputs and two outputs. It can be verified that either of the outputs can be activated (but not both), if and only if one of the inputs is activated. When both inputs are activated, both outputs can be activated. Using such a half crossover, a *crossover* can be constructed, see Figure 2(c). Signal splitters are displayed as a forking symbol; half crossovers are displayed as a plus symbol. The crossover contains two inputs and two outputs. It can be verified that the right output can be activated if and only if the left input is activated, and the top output can be activated if and only if the bottom input is activated. Thus, in order to prove a game PSPACE-hard, we need to construct gadgets that simulate variable setters, wires able to propagate and split the signal, and logic AND, OR and CHOICE gates.

For obtaining our complexity result, we consider a generalized version of the game. The CNF formula will be simulated on a $m \times n$ board, where both players have k pieces. Whether a natural generalization of the game would imply that the k pieces all have a strength in the interval $[1, 8]$ or a distinct strength from the interval $[1, k]$ is open for debate. In our gadgets all pieces have a strength in the interval $[1, 8]$, but trivial adjustments can change this to a distinct strength in the interval $[1, k]$. The original game board contains several properties, i.e., clustered water squares, narrow paths between the water, only traps around the dens, and symmetry. Which of these properties should be preserved on a generalized game board is also open for debate, however in our proof we took the liberty to freely use water squares and traps in the gadgets.

In the gadgets we use pieces with a strength in $\{2, 4, 5, 8\}$. Note that none of the pieces will have the ability to leap, neither will we use rats, so no pieces can move on water squares. The propagation of a signal in the circuit representation will be modelled as the movement of a white panther from one gadget to another. If the white panther can reach the output of the gadget corresponding to the output of the circuit, it will reach the black den and win the game. The black player, on the other hand, is given a piece that can move unopposed to the white den. However, it will start at a far distance from the white den, giving the white player the opportunity to reach the black den first, if possible. The black den will be protected by a black dog; it can only be reached by a white piece of equal or higher strength.

The gadgets are shown in Figure 3. Squares adjacent to the border not containing water, are called *entrance squares*. Entrance squares can correspond to either inputs or outputs of the gadgets, and will be addressed as such. A construction called the *gadget protector* will be connected to all entrance squares. It ensures that pieces can not enter a gadget through the outputs, and pieces can not leave a gadget through the inputs. Furthermore, it ensures that at most one piece can leave through each output. The gadget protector will be shown further on.

Lemma 2.1 *The construction in Figure 3(a) satisfies the same constraints as a logic AND gate, with the entrance squares at the bottom corresponding to the inputs and the entrance square at the top to the output.*



(a) AND (b) OR (c) FANOUT (d) CHOICE (e) VARIABLE

Figure 3: Dou Shou Qi gadgets.

Proof The white player can move a panther to the output square if and only if he arrives with a panther at both input squares. In order to pass black's elephant, the white player needs to sacrifice one of the panthers; the other piece can pass.

Lemma 2.2 *The construction in Figure 3(b) satisfies the same constraints as a logic OR gate, with the entrance squares at the bottom corresponding to the inputs and the entrance square at the top to the output.*

Proof Note that this gadget is almost the same as the AND gadget, the only difference is that an additional white panther is provided. The white player needs at least one panther arriving at the gadget. In order to pass the black elephant, one of the panthers must be sacrificed; another panther can pass. Whether the white player arrives with one or two panthers does not matter, only one can pass.

Lemma 2.3 *The construction in Figure 3(c) satisfies the same constraints as a signal splitter, with the entrance squares at the bottom corresponding to the input and the entrance squares at the top to the outputs.*

Proof The white panther in the gadget can not move before the black dog moves away from the traps, otherwise it will be captured. When another panther arrives, the black piece can be driven away from the traps and white has two panthers, which can each leave through a different output.

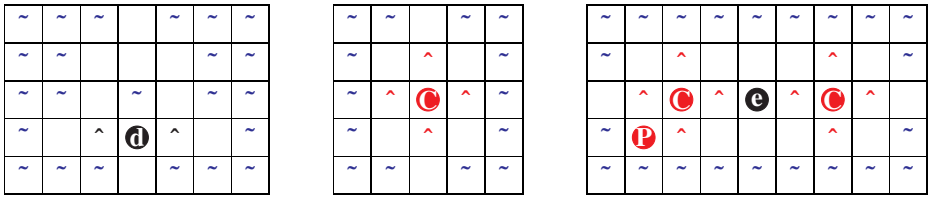
Lemma 2.4 *The construction in Figure 3(d) satisfies the same constraints as the described CHOICE gate: Upon arrival a panther can go either way, but not both. The entrance square at the bottom corresponds to the input and the entrance squares at the top to the output.*

Proof Once the white player moves a panther into the gadget, he can either move it through the left output or through the right output. Note that the gadget protector (shown further on) will ensure that this piece can not move through one of these entrances and move back later on.

Lemma 2.5 *The construction in Figure 3(e) satisfies the same constraints as a variable setter.*

Proof Like in the original G_{pos}(POS CNF) game, players take turns choosing a variable. At the moment a player choses a certain variable, he captures the opposing piece within the corresponding gadget. Variables chosen by the white player will thus allow a white panther to leave the gadget. Variables chosen by a black player will allow none of the pieces to leave the corresponding gadget.

In order to ensure that none of the gadgets allow actions that are not allowed in G_{pos}(POS CNF), all inputs and outputs are connected to a so-called gadget protector. The gadget protector is a chain of special constructions, shown in Figure 4. The construction shown in Figure 4(a) is a *black edge protector*. The white player can move a panther from bottom to top, but not the other way around. When a panther enters



(a) Black edge protector (b) White edge protector (c) Outflow protector

Figure 4: Dou Shou Qi supporting constructions for building the gadget protector.

the construction, the black dog will retreat behind either the left trap or the right trap, and the white panther can pass. When passed, the black dog moves back to its original position. The white panther can not move back, he would be captured upon entering the traps. The construction shown in Figure 4(b) is a *white edge protector*, it only allows white pieces to pass. Black pieces will be captured upon entering a trap. Note that these constraints do not apply when the opposing player attacks from both sides. We will show further on how to deal with this. The construction in Figure 4(c) is an outflow protector, with the entrance square left as input and the entrance square right as output. It ensures that upon arrival with either one or two panthers, only one can pass. This ensures that a player will never benefit from sending both pieces in a FANOUT gadget through the same output.

A chain of two white edge protectors, one black edge protector and another two white edge protectors is called a *one-way channel*. First, it ensures that no black piece can move through it. It will be captured upon entering a white edge protector. After a capture, the white cat can retake position, preventing black pieces from passing, regardless their number. Because it is always adjacent to another white edge protector, even an attack from both sides is useless. Second, it ensures that all black pieces within are unable to move out of the construction they started in, by the same argument. Finally, linking several one-way channels to each other ensures that white pieces can move through it in only one direction. White pieces that move in the opposite direction will be stopped at the black edge protector. Indeed, when having a piece at both the input and the output, the white player can enable its piece at the output to move back past the dog. However, in order to pass a number of subsequent black edge protectors, the white player needs an equal number of white panthers at the input to ensure such a passing. There can never be more than two white pieces at the input of a one-way channel, thus linking three one-way channels together prevents white pieces from moving into the wrong direction. A gadget protector is a chain of three one-way channels, one outflow protector and another three one-way channels.

Theorem 2.6 *Deciding the winner from a given Dou Shou Qi configuration is PSPACE-hard.*

Proof Reduction from G_{pos}(POS CNF). Given a positive CNF, we construct a corresponding Dou Shou Qi configuration. The number of pieces and the size of the game board are bounded by a polynomial over the number of variables and clauses in the CNF. The white player has a forced win if and only if he has a forced win on the original G_{pos}(POS CNF); otherwise the black player has a forced win. Note that there are no draws in G_{pos}(POS CNF), neither are there in the reduction. The gadget containing the white den must be chained to a black edge protector. This ensures that no other white pieces except the panthers can reach the black den. Black will not benefit from moving the dog into this gadget; the dog is still unable to stop a panther, regardless of the traps. The white player can move a piece into the black den if and only if he can set the corresponding CNF to true. The black player always has the potential to move a piece into the white den, but only after a certain amount of moves. If the corresponding CNF can be set to true, by that time the white player has already reached the black den.

If we apply a rule that imposes a polynomial upper bound on the number of moves, similar to the 50 moves rule used in chess, PSPACE-completeness could also be proven. Since this is a subjective matter, we refrain from this. Without such upper bound, Dou Shou Qi probably is in an even harder complexity class, such as EXPTIME or EXPSPACE.

3 Dou Shou Qi Engine

To get a feeling for the search complexity of a Dou Shou Qi game, we present some numbers. An average configuration allows for 20 legal moves (out of a maximum of 32). In theory a complete game tree of all possible games from the initial configuration can be constructed by recursively applying the rules of the game. From the initial configuration the number of leaves visited per ply can be found in Table 1. Assuming an average game length of 40 moves (80 plies), there are approximately 20^{80} possible games.

In this section we introduce a Dou Shou Qi (analyzing) engine, similar to a Chess engine which is used to search through the game tree given a certain configuration. The seminal 1950 paper by Shannon [11] lists the elements of a chess playing computer (also known as an engine), which are also applicable to a Dou Shou Qi engine as both games are similar. Usually, an engine consists of three parts: a move generator, which generates a set of legal moves given a configuration; an evaluation function (or utility function), which is able to assign a value to the leaf of the game tree, and a search algorithm to traverse the game tree. As evaluation function we use the method constructed in [1].

Table 1: The number of leaf nodes that are evaluated at a certain depth from the initial configuration. The performance was measured on an Intel i7-2600 with 16 GB RAM (no pruning).

ply	time	number of leaf nodes	evaluation	move
1	0.00	24	5	`Eb3`
2	0.00	576	0	`Eb3`
3	0.00	12,240	3	`Eb3`
4	0.06	260,100	0	`Eb3`
5	1.26	5,098,477	3	`Eb3`
6	23.46	99,860,517	0	`Eb3`
7	7:51.33	1,890,415,534	3	`Eb3`

In most chess playing engines today some form of the minimax algorithm is used. Here, one tries to minimize the possible loss for a worst case (maximum loss) scenario. The performance of the minimax algorithm can be improved (among many other methods), without affecting the result, by the use of alpha-beta pruning. Here, a branch is not further evaluated when at least one of the immediately following configurations proves to be worse (in terms of the evaluation function) than a previously examined move [5, 6]. In our case, using the same machine, we are able to search the game tree within the eight minutes to a depth of 14 plies. The aforementioned search methods operate on trees, while the actual search space is an acyclic graph. Configurations that have been considered before might be considered again by means of so-called transpositions. A reordered sequence of the same set of moves results in the same configuration. This is especially true for end game configurations where a few pieces can move around in many ways to form equal configurations. By storing evaluated configurations in memory we can omit expensive re-searches of the same configuration. Commonly, a hash table is used. Zobrist [14] introduced a hashing method for chess which can easily be extended to a more general case. The idea is to combine random numbers (64-bits) for each piece at each location by using the XOR operator. When using a set of carefully constructed random numbers the change of two different configurations that result in the same hash key can be ignored. Another nice property of this method is that a new key (after a move) can be constructed from the old key (current configuration). In the transposition table we store the hash key together with information about this configuration, i.e., the evaluated score (either exact or a bounding score in case of alpha-beta pruning), the depth the configuration was evaluated for, as shallower results are less trustworthy than deeper searches. The transposition table size is obviously smaller than all 64-bit possible hash keys. We map the hash key to an entry in the table by taking the values of the key modulo the table size as its index.

Our engine consists of the minimax algorithm with added alpha-beta pruning and augmented with a (large) transposition table using the Zobrist hashing method. This engine has proving its usefulness by performing a sanity check on the endgame tablebases presented in Section 4, which in turn can be used to improve the engine.

4 Retrograde Analysis

An endgame tablebase describes for every configuration including a certain number of pieces, precalculated information which side has a theoretical win. An endgame tablebase can be exploited in several ways. First, it can be incorporated in a Dou Shou Qi playing computer, potentially resulting in better play. In fact, an algorithm does not need to look for a line of moves that leads to a direct win, but it suffices to look for a line of moves that leads to a configuration in the tablebase which is known to be winning. Moreover, it can be mined for patterns, to gain novel insights, as we will see next. Endgame tablebases exist for many well-known games, such as Chess [12], Kriegspiel [2] and Checkers [9]. An endgame tablebase containing information about Checkers games containing at most ten pieces has been constructed by the authors of [9], eventually leading to a proof showing that by optimal play of both players, the game will result in a draw [10].

The technique we use for constructing this database is retrograde analysis, similar to the technique used by the author of [12]. All configurations are generated, and the configurations that are a terminal state, i.e., a win for white, a win for black or a stalemate, are marked as such. After that, we repeat the following procedure several times: For each configuration we evaluate all moves. If one of these moves leads to a configuration which is marked as a loss for the player to move, this is clearly a win, so we can mark it as such. If all these moves lead to a configuration which is marked as a win for the player to move, this is clearly a losing configuration, so we mark it as such. In all other cases we do not mark this configuration yet. This procedure is repeated until it does not lead to newly marked configurations. At that moment, we can mark all unmarked configurations as a draw, write all configurations to the database and terminate the algorithm. Note that we only need to write configurations with white to move to the database; configurations with black to move can be obtained from symmetry.

The resulting endgame tablebase is represented in a format that is readable for both human and machine. Each configuration is stored on a separate line. Each line contains a description of the configuration, the outcome by optimal play, a move yielding this result and the length of such a sequence. Optimal play is considered to be a shortest sequence of moves that lead to a win, or if the position is lost a longest sequence of moves that leads to a loss. We were able to generate endgame tablebases for configurations containing up to four pieces, but the program can be scaled easily to generate tablebases containing even more pieces. Some general statistics about the endgame tablebase are shown in Table 2. Each row summaries the number of configurations it contains, and the distribution of obtainable results for the player on turn. Furthermore, the longest sequence of moves that leads to a forced win for either player is displayed.

Table 2: Summary of the endgame tablebase up to four pieces.

pieces	positions	to move wins	to move loses	draws	longest sequence
2	160,068	82,852	64,501	12,715	34
3	54,354,684	30,297,857	23,369,820	687,007	67
4	9,685,020,510	5,468,841,129	4,001,236,829	214,942,568	117

The endgame database can be mined in search for interesting patterns. Although this is considered as the main focus of future work, an interesting result was already obtained. It has been suggested by some that this game often results in a draw, however our results show a surprisingly low ratio of draws. Furthermore, endgames consisting of two pieces with equal strength will never end in a draw under optimal play. There is a theoretical argument to support this claim. These endgames can be divided into two categories. On the one hand, there are the endgames which have a straightforward ending, e.g., one of the pieces can move unopposed to the opposing den or can capture the other piece right away. On the other hand, there are endgames which seem harder to assess, because both pieces yet have to pass each other in order to reach the opposing den, seemingly able to defend their own den. In this case, the Manhattan distance between the pieces is decisive; when this is an even number, the player to move can always win, otherwise the other player can win. In fact, by walking toward the opposing piece, it can be driven into a corner, an eventually be captured. This observation suggests that the notion of *parity* is important part in endgames. Note that tigers and lions are able to flip the parity, by leaping over the water.

5 Conclusions and Future Research

Our main contribution is a complexity proof, showing Dou Shou Qi is PSPACE-hard, which implies that it is an interesting game to study [4]. Furthermore, we created a playing engine and presented a method for constructing an endgame tablebase. Both can be used to gain novel insights in the game. As well it can be considered as a first step towards theoretically solving Dou Shou Qi in the same way Schaeffer *et al.* [9] solved Checkers. Finally, we have presented a theoretical proof that an endgame with two pieces of equal strength will not result in a draw.

Expanding the tablebase to more pieces is considered to be future work. More interesting patterns are still to been found in these tablebases, e.g., the impact of the parity on winning configurations. The fact that Dou Shou Qi could not be proven PSPACE-complete without the introduction of additional rules suggests that it is contained in a superset of PSPACE, possibly EXPTIME or EXPSPACE. Improving the complexity result to either one of these classes is also an interesting direction for future work.

References

[1] J. W. Burnett. Discovering and Searching Loosely Coupled Subproblems in Dou Shou Qi. Master's thesis, Tufts University, 2010.

[2] P. Ciancarini and G.P. Favini. Retrograde Analysis of Kriegspiel Endgames. In *IEEE Conference on Computational Intelligence and Games*, pages 411–418, 2010.

[3] E. D. Demaine and R. A. Hearn. Constraint Logic: A Uniform Framework for Modeling Computation as Games. In *IEEE Conference on Computational Complexity*, pages 149–162, 2008.

[4] R. A. Hearn. Amazons, Konane, and Cross Purposes are PSPACE-complete. In *Games of No Chance III, Proc. BIRS Workshop on Combinatorial Games*, pages 287–306, 2005.

[5] D. E. Knuth and R. W. Moore. An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, 6(4):293–326, 1976.

[6] J. Pearl. The Solution for the Branching Factor of the Alpha-Beta Pruning Algorithm and its Optimality. *Communications of the ACM*, 25(8):559–564, 1982.

[7] D. B. Pritchard and J. D. Beasley. *The Classified Encyclopedia of Chess Variants*. Beasley, 2007.

[8] T. J. Schaefer. On the Complexity of Some Two-Person Perfect-Information Games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.

[9] J. Schaeffer, Y. Björnsson, N. Burch, R. Lake, P. Lu, and S. Sutphen. Building the Checkers 10-Piece Endgame Databases. In *Advances in Computer Games*, pages 193–210. Springer, 2004.

[10] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers Is Solved. *Science*, 317(5844):1518–1522, 2007.

[11] C. E. Shannon and T. Hsu. Programming a Computer for Playing Chess. In *National IRE Convention*, 1950.

[12] K. Thompson. Retrograde Analysis of Certain Endgames. *ICCA Journal*, 9(3):131–139, 1986.

[13] J. N. van Rijn. Playing Games: The complexity of Klondike, Mahjong, Nonograms and Animal Chess. Master's thesis, Leiden University, 2012.

[14] A. L. Zobrist. A New Hashing Method with Application for Game Playing. *ICCA Journal*, 13(2):69–73, 1990.