Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 67 (2007) 105–124
Science of Computer Programming
www.elsevier.com/locate/scico

Solving games
Dependence of applicable solving procedures

M.J.H. Heule[a,*], L.J.M. Rothkrantz[b]

[a] *Department of Software Technology, Faculty of Electrical Engineering, Mathematics and Computer Sciences, Delft University of Technology, Netherlands*

[b] *Department of Mediamatica, Faculty of Electrical Engineering, Mathematics and Computer Sciences, Delft University of Technology, Netherlands*

Received 1 October 2005; received in revised form 7 July 2006; accepted 21 January 2007
Available online 5 April 2007

Abstract

We introduce an alternative concept to determine the solvability of two-player games with perfect information. This concept — based on games currently solved — claims that the applicable solving procedures have a significant influence on the solvability of games. This contrasts with current views that suggest that solvability is related to state-space and game-tree complexity. Twenty articles on this topic are discussed, including those that describe the currently obtained solutions. Results include a description of the available solving procedures as well as an overview of essential techniques from the past. Besides well-known classic games, the solvability of popular and recent games zèrtz, dvonn, and yinsh are analyzed. We conclude that our proposed concept could determine the solvability of games more accurately. Based on our concept, we expect that new solving techniques are required to obtain solutions for unsolved games.

Keywords: 2-player board games; Solving procedures; Solvability

1. Introduction

Since the beginning of artificial intelligence, strategic games have captured the imagination of the professionals in this field. Two separate lines of thinking can be pointed out: Can artificial intelligence outperform the human masters in the game? Secondly, can it determine the game-theoretic value of a game when all participants play optimally? This value indicates whether a game is won or lost from the perspective of the player who makes the first move. The methods to reach these goals can differ substantially.

In this paper, we will focus predominantly on solved games — those games for which this game-theoretic value is known — and on the solving procedures used to obtain these values. Why are some games solved, while others are beyond our reach? We will show that characteristics of games and the available solving procedures have a profound influence on their solvability. Moreover, we claim that the extent to which four essential solving techniques are applicable provide some answers to this question.

* Corresponding author. Tel.: +31 (0) 15 278 7263; fax: +31 (0) 15 278 6632.
E-mail addresses: marijn@heule.nl (M.J.H. Heule), l.j.m.rothkrantz@tudelft.nl (L.J.M. Rothkrantz).

0167-6423/$ - see front matter
doi:10.1016/j.scico.2007.01.004

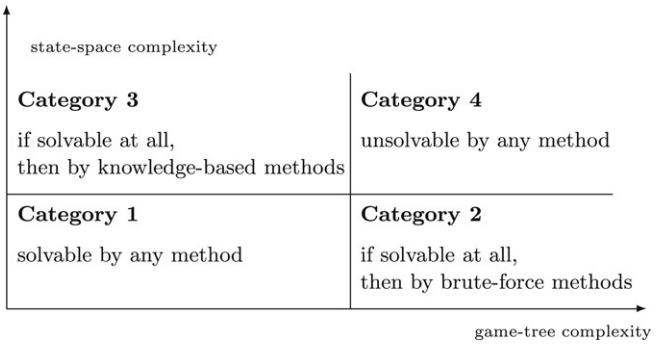

Fig. 1. A double dichotomy of the game space.

In a survey paper by Van den Herik et al. [10], the solvability of games is a central theme. According to this paper, solvability is related to the *state-space complexity* and *game-tree complexity* of games. The state-space complexity is defined as the number of legal game positions obtainable from the initial position of the game. The game-tree complexity is defined as the number of leaf nodes in the solution search tree of the initial position of the game. A dichotomy applied to each dimension roughly yields four categories following from the games' state-space complexity being low or high and the games' game-tree complexity being low or high. The classification of [10] is shown in Fig. 1.

Although the figure above seems plausible, its does not clarify why some two-player games are currently solved. For instance, let us look at the first solved game — qubic (1980) [13] — and the most recently solved game — awari (2003) [15]. According to Allis [1], both the state-space and the game-tree complexity of the latter are smaller than those of the former. These facts could hardly be explained by Fig. 1.

Here, we will suggest an alternative classification concept. Let the *solution size* refer to the number of positions for which the optimal move/strategy must be stored in a certificate proving the solvability of a game. This extends the notion of *decision complexity* — the number of decisions that is required to store a solution [3] — in the sense that a single strategy "covers" multiple desicions/moves. So, the solutions size is smaller or equal to the decision complexity. Solving procedures could be used to reduce the the size of such a certificate or the computational costs to obtain it. This notion is central in this paper: *The solvability of a game depends on whether the solving procedures could be applied to reduce the solution size to a reasonable complexity.* With "reasonable" we mean that the current computer hardware is sufficient to compute certificates of that complexity. Of course, due to the ongoing evolution of computer hardware this threshold increases through the years. When no procedures are applicable for a certain game, it can only be solved when its state-space or game-tree is below a reasonable complexity, say 10^{10} and 10^{20} respectively. Solving such a game can then be achieved using elementary solving procedures that are applicable to all games.

This classification concept differs in two ways from the classification concept based on state-space and game-tree complexity. First, this alternative approach does not make a distinction between knowledge-based methods and brute-force methods. Secondly, no game could be labeled as unsolvable by any method. An alternative view on game space, with this concept in mind, is shown in Fig. 2. Notice that the first three categories appear much smaller than in Fig. 1. This is done on purpose; the thresholds are much smaller than the borders of the categories that Van den Herik et al. [10] had in mind.

Using this classification of *dependence on applicable solving procedures*, we could explain past results in greater detail. However, it is not ideal: it takes time and effort to find out which procedures are applicable and to what extent these procedures could reduce the solution size. Beside this concept, we will provide an overview of various solving techniques that have been successful in the past — the proven formulas — and merely touch on some that might (or might not) be useful, since these are infinite in number and of unknown use. . . .

2. Related work

2.1. Preliminaries

In this paper, we restricted ourselves to zero-sum two-player games with perfect information. This excludes games like backgammon, in which chance plays a major part stratego and implies imperfect information. This restriction also excludes rubik's cube — which is only played by one player — and monopoly, which involves negotiation. Most

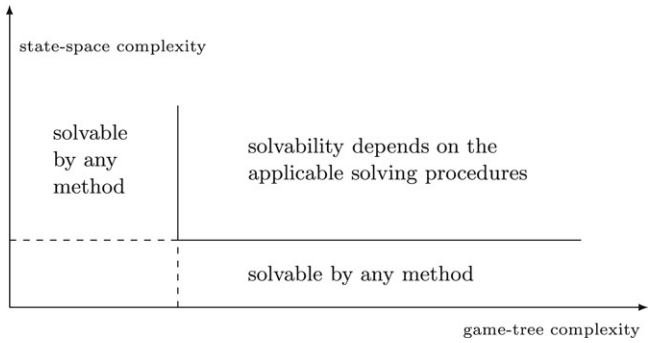

Fig. 2. An alternative view on the game space.

of these restrictions follow directly from the focus on the game-theoretic value: The game-theoretic value cannot be computed for games in which *chance*, *negotiation* or *imperfect information* are an important factor of the outcome. For example: the game-theoretic value of one-player games is trivial: It is always a win for the first player! The exclusion of these games is based on their substantial difference to two-player games.

Games of which the game-theoretic value is known are labeled *solved*. Three degrees of solutions have been defined by Allis [1]. Lincke [11] distinguishes these three solution categories as follows:

> **ultra-weakly solved** A game is called *ultra-weakly solved* if only the value of the start position is known. This corresponds to the general definition of a solved search problem. For example, there is well-known proof (John Nash, 1949) that the game of hex on a $n \times n$ board is a win for the first player for any n. So far, winning strategies are only known for $n \leq 9$ [21].
> **weakly solved** A game is called *weakly solved* if the value of the start position is known and a strategy is known which guarantees that the first player can achieve that value. This means that for example, if a game is known to be a draw, then the first player will never lose. If the opponent makes a mistake, the first player does not necessarily know a strategy which achieves the win.
> **strongly solved** A game is called *strongly solved* if the best move can be found (within reasonable time) from every position in the game.

The most interesting part of solving games lies less in obtaining the game-theoretic value, yet more in determining the winning strategy. Therefore, this paper considers only those procedures that weakly or strongly solve games.

2.2. Solved games and their complexities and solution sizes

What determines the solvability of a game? In their latest overview article on solving games [10], van den Herik et al. (2002) make an attempt to answer this question. As stated in the introduction, this article suggests that *state-space complexity* and *game-tree complexity* provide an answer to this question. Currently, eight two-player games[1] with perfect information have been solved. Table 1 shows the estimated values of the state-space and game-tree complexity by Allis [1] of these games, as well as the solution size of the first certificate proving the game-theoretic value. This concept of classification by the state-space and game-tree complexity appears logical, but the solutions of games played in the past do not correspond with this chart. Practically all past solutions show some friction in determining the solvability by the state-space and game-tree complexity.

Let's look at two examples in detail:

(1) Qubic is the first non-trivial game that was solved (1980) [13]. Apparently, this is an easy game to solve. However, classifying the game according to the statistics of the state-space and game-tree complexity does not give that impression: of the eight games that have been solved, five have a much lower state-space complexity. Four out of these five also have a lower game-tree complexity.

[1] This report describes eight games that have been solved. There are also other ones — often closely related to a solved one. Their removal does not change the general view.

Table 1
Solved games ordered by year of solving

Nr.	Year	Game	Search-space complexity	Game-tree complexity	Solution size	Reference
①	1980	qubic	10^{30}	10^{34}	2.929	[1,13]
②	1989	connect-four	10^{14}	10^{21}	_[a]	[1,19]
③	1994	go-moku	10^{105}	10^{70}	138.790	[1,2]
④	1996	nine men's morris	10^{10}	10^{50}	7.76×10^{9}	[1,8]
⑤	1996	pentominoes	10^{12}	10^{18}	_[a]	[12]
⑥	2000	domineering	10^{15}	10^{27}	4.42×10^{8}	[1,7]
⑦	2001	renju	10^{105}	10^{70}	813.674	[1,20]
⑧	2003	awari	10^{12}	10^{32}	8.89×10^{11}	[1,15]

[a] The reference describing the solvability does not provide the solution size.

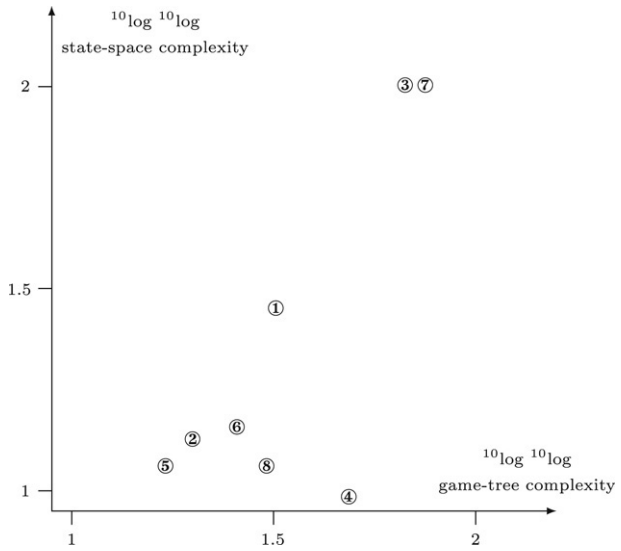

Fig. 3. Approximate positions of games in the game space using their sequence number of solving date.

(2) Awari has a very low game-tree complexity, and holds the lowest state-space complexity after nine men's morris. It should therefore be labeled as "solvable by any method". However, the game had not been solved in 2002, when Van den Herik et al. wrote their article! Currently, it has been solved, but only by brute force.

Probably the best illustration of the flaws of this classification concept is a game space plot (Fig. 3). This plot depicts all possible solutions, positioning them in the sequence of the moments they were solved. To dampen the exponential nature of both complexities, the measurements along the axes are given in $^{10}\log\ ^{10}\log$ format. The resulting image does not clearly reveal any logical relations.

Moreover in general, we see that only those games are solved whose solution size is less than 10^{12}. Also, the first solved game appears to have the smallest certificate proving solvability, while the latest solved game has by far the largest solution size. Moreover, there appears no single correlation between the size of the solution and its search-space or game-tree complexity.

3. Solving procedures

In this section, we will discuss the available solving procedures for two-player games with perfect information, and how these procedures could be used to reduce the size of the solution or the computational costs to obtain it. For this purpose, 20 articles were reviewed. An overview of the solving procedures mentioned in these articles is provided in Appendix A. In their article, Van den Herik et al. [10] explicitly define six procedures to solve games: *retrograde analysis*, *enhanced transposition table methods*, *threat-space search and λ-search*, *proof-number search*, *depth-first*

proof-number search, and *pattern search*. These procedures have been used as the basis to define five categories: *retrograde analysis*, *transposition tables*, *game-tree search procedures* (this category includes proof-number search and depth-first proof-number search), *winning threat-sequence procedures* (which includes threat-space search and λ-search), and *winning pattern procedures* (which includes pattern search).

3.1. Retrograde analysis

Retrograde analysis is a method in which, for each position of some specific game or endgame, the optimal moves towards the best reachable goal is stored. For instance, in chess, assuming perfect counterplay, the number of moves to be played by the stronger side up to mate or conversion is stored. Checkers databases sometimes only contain an indication of won, drawn, or lost per position. A database is constructed by starting in terminal positions and then working backwards. Once constructed, perfect play is guaranteed: the stronger side chooses a move with the shortest distance-to-mate (or to conversion) and the weaker side opts for moves with the longest distance-to-mate (or to conversion) [10].

Retrograde analysis has proven to be a very powerful technique for solving games. This technique is not really focused on reducing the solution size, but in the construction of a certificate piece by piece. This makes it possible to drastically reduce the computational costs. Important contributions of retrograde analysis are solving nine mens morris, checkers end-games, chess end-games, draughts end-games, and recently awari. In the near future, it is expected that checkers will be weakly solved from the initial position by the use of elaborate end-game databases. The method will be similar to the method used to solve nine mens morris [8]. The achievements of retrograde analysis have been made possible by its many enhancements. For a successful application of retrograde analysis, one should consider the following improvements:

Decomposition of the database: By dividing the database into multiple smaller sub-databases, the computation cost to obtain the whole database decreases drastically. Furthermore, the results can be categorized more effectively to retrieve a position. Decomposition can be achieved by splitting the database based on the number of pieces, the different kinds of material, and the leading rank [16].

For instance, using a combination of these concepts, the whole database of checkers (backwards to ten pieces) with a total of 3^{13} entries is reduced to slices with a maximum of 1.7×10^{11} entries [16]. However, these kinds of reductions are not applicable to all games: in games like awari, all the materials are identical, and it is not possible to identify the positions by leading rank.

Compression of the database: Various articles on retrograde analysis discuss compressing the database. For instance, multiple positions could be stored in one byte. A byte can store 2^8 ($= 256$) distinct values. For most game positions, only three possible values are needed: win, draw or, loss. Storing five positions ($3^5 < 256$) in one byte results in a compression by a factor of five compared to elementary storage methods. This could even be extended (see [16]).

Compression to this extent is harder to achieve for awari, which requires storage of the optimal difference in pieces instead of the win, loss, or draw values. Yet it is still possible: the algorithm described in [11] reduces the space needed to store a position of awari to only two bits.

Parallel retrograde analysis: A successful implementation of the parallel retrograde analysis is introduced by Romein and Bal [15]. Using this parallel algorithm, they compute all legal positions of awari — solving it strongly. Essential to this algorithm is that the work is migrated to other processors instead of shipping data.

Neglecting unlikely positions: Not all legal positions contribute to (weakly) solving a game. The combination of a game-tree search with retrograde analysis makes it possible to neglect unlikely positions. The game nine mens morris is solved using this technique [8]: Of all m–n piece end-game databases (m ranging from 3 to 9, n ranging from 3 to m), only the 9–9, 9–8, and 8–8 databases are used for determining the game-theoretic value. This is based on the low likeliness that one player loses two pieces during the opening phase of the game. If such a situation occurred, the opponent would likely win.

This idea of negligence of unlikely positions is also used to acquire a rough estimation of the upper bound of state-space needed to weakly solve checkers [16]. This provides a more accurate estimation of the effort to solve checkers than the state-space or game-tree complexity.

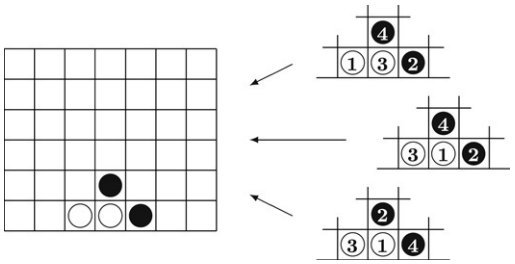

Fig. 4. Three opening sequences result in an identical position

3.2. Transposition tables

While solving a game, many identical positions will occur within the game-tree. Fig. 4 below shows a connect-four position after four moves that follows from three different opening sequences. When the game-theoretic value is known for one of these sequences, this value is also known for the others. Storing this value could prevent incurring additional costs. In a solution of a game, one table entry could be used to cover multiple positions. Therefore, if transposition tables are applicable, they could significantly reduce the size of a solution.

Due to memory issues, not all positions could be solved in a transposition table. Therefore, most transposition tables are implemented using a hashing method [6]. The following enhancements for transposition tables are described in the articles we review:

Table information: The amount of information that is stored in a transposition table per entry should be as small as possible. This way, more entries can be stored using the same amount of memory. For chess, at least the five elements should be in the transposition table [6]. Most other games require fewer elements for storing in the table: For domineering, only positions whose game-theoretic value are known are stored in the table [7]. Therefore, only a single bit-element is required for each position, describing whether it is won or lost.

Replacement Schemes: Since not all possible positions can be stored in a transposition table, an effective strategy is needed to replace collisions. These strategies are called replacement schemes. Two articles [6,7] offer seven replacement schemes. Both describe experiments to determine the performance of the various schemes. The first uses chess positions as a testbed, while the other focuses on their ability to weakly solve the game of domineering. Both articles conclude that the two-level table replacement schemes — so each entry has two positions in the table — perform best.

Table size: Two experiments concern the size of the transposition table. The first one [6] uses chess mid-game positions for the experiment and concludes: as table size increases, the number of nodes searched tends to stabilize. In other words, at some point, possibly before 1024K in our case, no significant gains may be expected by increasing table size.

The second experiment [7], by the same authors, uses domineering positions. Apparently, they were working towards the same conclusion, because only one table was larger than 1024K entries. In contrast with chess mid-games, this larger table showed a significant performance gain compared to the 1024K entries table. It remains a mystery why no larger tables were used in this experiment.

Another article on domineering [4] uses a table with 8192K entries. Under the assumption that the evaluation function without enhancements in [4] is identical to the one described in [7], it may be concluded that this elaborate table reduces nodes up to 60%. In conclusion, the optimal table size is likely game dependant.

Generalized positions: Some pieces or squares might not contribute to the game-theoretic value or the best move — depending on what information is stored in the transposition table. This could mostly occur in the end phase of a game. These pieces or squares could be labeled as *indifferent* [19]. Positions with indifferent pieces are called *generalized positions*. Multiple positions could be mapped on the same generalized position. An example is shown in Fig. 5.

Symmetry: In practically all games, symmetric positions occur. The amount of symmetries varies per game: Awari has only one symmetry (rotation by 180°) [15], while qubic has 192 symmetries [13]. When using these symmetries

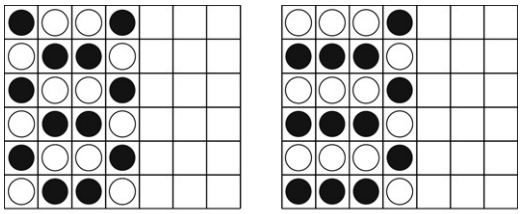

Diagram I Diagram II

Fig. 5. Two connect-four diagrams that could be mapped to the same generalized position: all pieces in the first two columns of both diagrams could not contribute to a winning line, and therefore not to the game-theoretic value. These pieces could thus be labeled as indifferent. All other pieces are equal.

in the opening phase of qubic, the possibilities of the first three moves can be reduced from $64 \times 63 \times 62 = 249\,984$ to only seven. By converting a position in such a way that all the symmetric positions are mapped on the same position, only a check is required to see whether this converted one exists in the transposition table.

3.3. Game-tree search procedures

A game-tree consists of all possible positions that could occur while playing the game. The initial position is the root of the tree and all terminal positions (where one of the players has won, or the game has reached a draw) are the leaves. The size of the game-tree is usually estimated by powering the average number of moves per position and the average number of turns. For othello, these values have been estimated to be 10 and 58, respectively [1]. The estimated game-tree size would thus be 10^{58}.

Game-tree search deals with an efficient examination of the tree in order to achieve a desired outcome. Many procedures have been developed to reach a goal as efficiently as possible. There are many uncertainties that need to be handled by heuristics. An example: If the goal is to prove that WHITE could always win from its initial position, one has to prove that there exists a move for WHITE, so that for every move of BLACK, there exists another move for WHITE, and for any possible response from BLACK, there is a move for WHITE by which WHITE eventually wins.

But how to reach this goal efficiently? First, this procedure will always try to select a winning move for WHITE. This way, it examines only one branch of the tree: every position where WHITE is to move. Second, if WHITE can choose from multiple winning moves, this procedure strives to select the one which will result in the smallest sub-tree. This will reduce the size of the solution. However, to fulfill these wishes, an oracle of Delphian proportions is required. Since we don't have one at our disposal, even more guesses have to be made to search efficiently: if a WHITE move would accidentally result in a draw or loss, one would want to detect this error as soon as possible. Therefore, of all the BLACK moves at a given position, we need to look for the best BLACK move with the smallest possible sub-tree. In essence, game-tree search procedures are concerned with estimations of the best move and the smallest sub-tree.

A large number of game-tree search procedures have been developed. Yet only three of them have been actually used to solve games: *α–β search* [1], *conspiracy-number search* [1], and *proof-number search* [17]. The usefulness of most other game-tree search procedures is as yet unknown: if they are applied to games at all, they are used in chess or go. Since both of these games are far from weakly solved, their performance at these games is no indication of their solving capabilities. Two procedures are actually tested on two other games: *PN** and *PDS* [17].

It would be interesting to see the exact reduction of the solution size that could be realized by the various game-tree search procedures. Moreover, to what extent do these procedures approximate the minimum size of the game-tree?

3.4. Winning threat-sequence procedures

All games involve threats like sudden death (e.g. check in chess) or the loss of pieces. Especially, sudden death threats can drastically reduce the game-tree search and the solution size, since the opponent is heavily hampered in his/ her mobility: he is forced to reset his/ her priorities and first counter the threat. We refer to a *winning threat-sequence* as a sequence of threats to which the opponent has a limited — and diminishing — set of replies. This way, a win could be forced. Because the opponent is limited in his/ her mobility, only a sequence of moves — instead of a sub-tree — can be stored with a position in a solution certificate. Fig. 6 shows an example of a winning-threat sequence

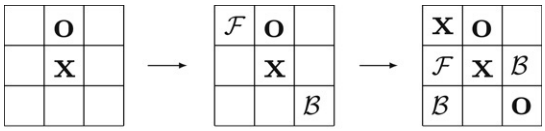

Fig. 6. An example of a winning threat-sequence in tic-tac-toe. Player **X**'s moves at $\mathcal{F}$ in positions *II* and *III* force player **O** to block at $\mathcal{B}$. However, in position *III*, player **O** has to block two squares. Since this is impossible, player **O** loses.

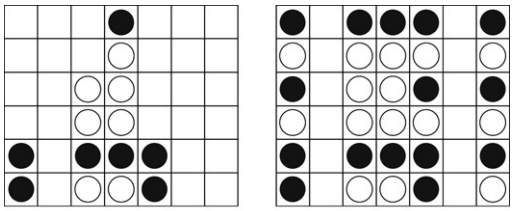

Fig. 7. Two connect-four diagrams. WHITE to move loses.

for tic-tac-toe. Procedures which investigate the existence of such a sequence in a certain position are called *winning threat-sequence procedures*.

Three procedures concern winning threat-sequences: *threat-sequence search*, *threat-space search*, and *lambda search*. The first two terms are used for identical procedures in some papers, but in this paper, we make a clear distinction. The following procedures are associated with these terms:

Threat-sequence search: Threat-sequence search is a procedure which explores all possible combinations of threats until no new threats occur. If one exists, this search procedure will always find a winning threat-sequence, although this may be very time-consuming. Threat-sequence search was used to solve qubic [13].

Threat-space search: The threat-space search procedure is a more human-like approach of finding a winning threat-sequence. Not all possible combinations of threats are explored, but only the ones that will probably result in new threats. The costs of this procedure are significantly less than the threat-sequence search. However, in some cases a winning threat-sequence is not found. A general description of this procedure can be found in [1], and a specialized version of this procedure for go-moku can be found in [2].

Lambda search: Lambda search is a procedure that looks further than a mere search for a winning threat sequence. It looks for a winning threat sequence if the opponent passes on the next turn. This search technique is only applicable when passing is allowed in a game, or when zugzwang is not a motive. This does not mean that the winning threat-sequence only occurs if the opponent passes: a winning threat sequence will occur whenever the opponent does not counter it. If no winning threat-sequence exists at the current turn, Lambda search is very useful to determine a relevant next move, and to find a winning threat-sequence in the next turn [18].

3.5. Winning pattern procedures

Many games become easier to solve during their progress. This makes it possible to define a strategy for a guaranteed win. Once the existence of such a strategy becomes evident, further searching in the game-tree becomes superfluous; the game-theoretic value for that position is already known. Since most of these strategies are formalized patterns, such a strategy is referred to as a *winning pattern procedure*.

An example: Consider the connect-four diagrams in Fig. 7. It is obvious that WHITE will lose in Diagram II: WHITE has to play either in column two or six, while BLACK will win by playing into the same column. Diagram I also shows a clear win for BLACK. BLACK has an easy strategy which will lead to Diagram II: BLACK always has to play into the same column as WHITE. This will force WHITE to move into a column it would rather not use.

These winning pattern procedures could be used to reduce the solution size: instead of storing a subtree of positions showing that there exists a winning strategy that guarantees a victory, only the pattern could be stored with a position

in the solution certificate. Six procedures discussed in the reviewed articles are not mentioned yet. These could be considered — in some sense — as *winning patterns*. These procedures are:

① Strategic rules for connect-four [19]
② Both evaluation functions for domineering [4,7]
③ Decomposition patterns for hex [14,21]
④ Partition search [9]
⑤ Pattern search (for hex) [14]
⑥ Go patterns generated by retrograde analysis [5]

These six procedures are grouped in two categories: *hand-made patterns* and *computed patterns*.

Hand-made patterns: The procedures of this category are ①, ②, and ③. All these patterns are hand-made by human experts, but they also share another property: They have been essential to weakly solve some games.

– A combination of *strategic rules* ①, conspiracy-number search and enhanced transposition tables were sufficient to weakly solve connect-four.
– The development of *local patterns* for hex ② made it possible to solve it for up to a 9×9 board. Other techniques were only able to weakly solve the game for up to the 7×7 board.
– The substantial reduction of the game-tree (in terms of nodes) by the *winning strategy functions* ③, was essential to solve large boards of domineering.

A disadvantage of these hand-made patterns is that they are game-specific. They could not be easily applied to other games.

Computed patterns: The procedures ④, ⑤, and ⑥ belong to this category. In contrast to the hand-made patterns, none of these procedures made a contribution to any (weak) solution of a game. However, this does not mean they are not useful:

– *Partition search* ④ significantly improved the performance of the game-tree search in the game of bridge (imperfect information). The procedure is promising, but it is not clear as to which other games it could be successfully applied to.
– *Pattern search* ⑤ has not been tested in its ability to weakly solve hex. It would be interesting to see whether it could compete with the local pattern procedure ③. On larger boards it may not longer be possible to construct all the patterns by hand. An automated generation of patterns is probably required to solve hex on the large sized boards. This might be realized by either pattern search or a modification of local patterns.
– The study of the *computer-generated patterns* for go ⑥ is too small to make some conclusions. Further experiments are required to indicate the usefulness of this method.

3.6. Essential solving procedures

In the introduction, we presented a new concept for determining the solvability of games. This concept ascribes a substantial influence to solving procedures on the solvability of games — depending on their application to reduce the solution size or the computational costs to obtain a solution. Besides the solution of pentominoes, essential solving procedures could be appointed for each solution discussed in the articles. Table 2 shows the essential solving procedures ordered by games. The exception of pentominoes can be explained by its very low game-tree complexity. Since it is below the estimated game-tree threshold of this concept (see Fig. 2), it was solved by elementary procedures. The main reason why this game was solved quite late (1996) is probably due to its lack of popularity.

Looking at the essential solving procedures, two large gaps appear:

(1) None of the game-tree search procedures are mentioned in Table 2. This does not mean that they were not applied to reduce the solution size: although the connect-four solution applied conspiracy-number search and the solution of go-moku used proof-number search, neither of these articles related to the solutions give the impression that these procedures were essential.

Table 2
Games with their essential solving procedures

Game	Essential solving procedure(s)	Reference
awari	Retrograde analysis	[15]
checkers end-games	Retrograde analysis	[16]
connect-four	Hand-made patterns, transposition tables	[19]
domineering	Hand-made patterns, transposition tables	[4,7]
go-moku	Threat-space search	[2]
hex (9×9)	Hand-made patterns	[22]
nine men's morris	Retrograde analysis, transposition tables	[8]
pentominoes	None	[12]
qubic	Threat-sequence search	[13]
renju	Threat-sequence search	[20]

(2) As stated in the previous section, computer generated patterns have not yet contributed to the solution of any game. This sharply contrasts with hand-made patterns, which were all crucial to some solutions.

4. Results

The solving procedures discussed above — and thereby the reduction of the solution size — cannot just be applied to any game. Application depends largely on the characteristics of a game. Four solving techniques appeared essential to solve games in the past (see Section 3.6). In this section, we present four game characteristics analogous to these four essential solving techniques. We conclude with an overview of these characteristics and how they could be used to determine solvability.

4.1. Convergent endgames

Retrograde analysis is designed for games with *convergent endgames* — endgames for which the size of the state-space decreases as the game progresses. Since only nine men's morris, awari, checkers, and chess have convergent endgames, this technique is only applicable to them. To strongly solve a game by applying retrograde analysis, it is required to store the complete legal state-space. Current computers are only capable of storing this data for nine-men's morris and awari. In the near future, checkers may be added to that list. However, the complete state-space of chess is far too extensive to be strongly solved by merely applying this technique.

4.2. Equivalent positions

Many different opening sequences result in *equivalent* — identical, symmetric, or generalized — positions. The number of different opening sequences resulting in the same equivalent position depends on the symmetries and material of a game. Games as qubic, connect-four, go-moku, domineering, renju, hex, and go have multiple symmetries and consist only of two kinds of material. Therefore, many equivalent positions occur in the search space. The same holds for awari: although it has only one symmetry, many equivalent positions occur, asonly one kind of material is used. Due to the diversity of the material in both pentominoes and chess, equivalent positions will occur relatively less frequent compared to the other games.

Mostly, equivalent positions occur for those positions that result after the same number of turns. Therefore, the depth of the search-tree influences the usefulness of transposition tables: a very deep game-tree requires a very large transposition table. So, since go-moku has a smaller game-tree depth compared to renju, transposition tables applied to the former could be smaller than those applied to the latter. Likewise, the immense game-tree depth of go will require an enormous transposition table.

4.3. Sudden death threats

For many games, there is the threat of an instant win by the opponent. We refer to such a threat as a *sudden death* threat. Most of the time — except for many chess positions — the player is forced to make the only move that

Table 3
Degree of possible application of solving procedures to various games

Year	Game	Retrograde analysis	Transposition tables	Threat sequences	Pattern strategies
1980	qubic		■■■	■■■	■■
1989	connect-four		■■■	■■	■■■
1994	go-moku		■■■	■■■	■■
1996	nine men's morris	■■■	■■		■■
1996	pentominoes				
2000	domineering		■■■		■■■
2001	renju		■■	■■	■■
2003	awari	■■■	■		
>2004	checkers	■■	■■		■
>2004	chess	■		■	■
>2004	hex 15×15		■■		■■
>2004	go		■		■

cancels this ultimate threat. Using winning threat sequences (see Section 3.4) could force sudden death by eliminating the freedom of the opponent to make his/ her own move.

The number of turns that are required in a game before one could apply a winning threat sequence is a profound influence on the effectiveness of this technique to strongly solve that game. In games like qubic and go-moku, this number is on average very low, say 1–10. For games like connect-four and renju, this number on average is a bit higher, say 11–20, and for chess it is $\gg 20$, rendering this technique barely applicable.

4.4. Local endgames

A focus on the complete space of the board is not always required in order to determine a winning strategy in the end phase. We refer to such an endgame — in which the majority of the board could be neglected in a winning strategy — as a *local endgame*. Local endgames are frequently generalizations of a few winning strategies. The frequency of local endgames (compared to global endgames) is strongly related to the application of winning pattern strategies: since only a subspace — which could be located anywhere — of the playing field is required for a winning strategy, pattern strategies as described in Section 3.5 could be applied.

In awari, checkers, and go, material progress can sometimes be made locally, but winning strategies generally have a global nature. Although chess has sudden death treats, winning strategies are rarely local.

4.5. Solvability

How could one determine the solvability of games based on their characteristics and corresponding solving procedures? Table 3 summarizes the extent to which these characteristics — and hence the analogous solving procedures — are applicable to various games. The games are chronologically ordered by date of solving. The application degree is in accordance with the descriptions above. Looking at this table, one can notice that the first three solved games have the same scores in all categories. Apparently, games with these characteristics are easy to solve.

This table shows that the order — based on the solution date — is highly related to the sum of the black squares — the degree of applications. As stated before, the exception of pentominoes can be explained by its lack of popularity.

The games checkers and hex will probably be weakly solved within this decade: both perform relatively well in two categories; with some effort and with the ongoing growth of computational power in mind, solutions could be expected relatively soon. However, for chess and go, there is no hope that either will be strongly solved within the coming decades, since none of the existing powerful solving procedures can determine the game-theoretical value. Here, new solving procedures are required.

5. Gipf

Games like chess, checkers, go, and awari have existed for ages. Yet, recently, Kris Burm has developed a series of six two-player board games with perfect information — gipf, tamsk, zèrtz, dvonn, yinsh, and pünct — which

have won various awards.[2] These games provide great challenges in the field of artificial intelligence, since these games will probably be very hard to solve.

These games have many interesting characteristics in which they differ from the classic games. Four examples: (1) In tamsk the material is time-based, with each piece having its own hour-glass. This results in multi-dimensional time pressure — in contrast to playing a classic game with a single clock; (2) In dvonn the players first have to determine the starting position in a turn-based fashion — yielding billions of possible starting positions; (3) In most games, each player has its own kind of material. However, in zèrtz all the material is shared between both players. (4) In classic n-in-a-row games, the player who first constructs a row of his/ her color, wins. In contrast: winning yinsh requires a player to construct *three* rows.

This section will focus exclusively on the solvability challenges of zèrtz, dvonn, and yinsh, because these games are by far the most popular in this series.[3] We will discuss here three characteristics of these games in which they differ from the classics. To handle the difficulties that arise from these characteristics will probably require new solving procedures. The rules of zèrtz, dvonn, and yinsh are provided in Appendices B, C and D, respectively.

5.1. Billions of starting positions

Classic two-player board games have a fixed starting position. This feature is very useful for computer players as they can use an opening book to guarantee a strong initial strategy. Obviously, a game like chess would be even harder to solve if it were to have multiple starting positions.

The games dvonn and yinsh have multiple starting positions. These are set out in a *placement phase*: before the players move their pieces, they first determine a starting position by placing the pieces on the board in a turn based fashion.

Due to this placement phase, the number of initial positions is enormous: playing dvonn requires first the placing of three dvonn pieces, followed by the alternate placing of the black and white pieces. This placement phase can result in $\frac{49!}{3!23!23!} \approx 1.5 \times 10^{17}$ positions. Even when we consider the symmetries of the board, approximately 4×10^{16} different positions remain feasible.

For yinsh, the number of positions is slightly smaller, but still very large: after alternately placing the black and white rings on the board, $\frac{85!}{5!5!75!} \approx 7.9 \times 10^{14}$ positions are possible. When considering the symmetries, approximately 8×10^{13} different starting positions remain at hand.

Because dvonn and yinsh have billions of starting positions, they are probably extremely hard to solve: even if the game-theoretic value of a single starting position can be computed, it would still be very complicated to determine this value for the initial position of the placement phase.

5.2. No pain, no gain

Threats are important in solving many games (as discussed in Section 3.4). The number of moves of the opponent can be greatly reduced by threatening: to create a row for example, or to capture some material. In zèrtz and yinsh, these threats do exist, but their application also weakens the player's own position. Therefore, threat sequence procedures could contribute to only a minor reduction of the game-tree for these games.

Capturing marbles in zèrtz is rarely for free. Generally, one has to sacrifice some marbles to the opponent in order to capture marbles. Essential to winning the game is a clear estimation of the relative importance between sacrificed and captured marbles: a player needs to avoid a situation where the opponent owes his/ her victory to sacrificed marbles.

Compared to all other make-a-row games, yinsh has an important difference: whereas other games require only one row to claim victory, yinsh requires three. Therefore, a sudden death threat rarely occurs in yinsh.[4] Moreover, by making a row, the player undermines his/ her own position: (1) A ring of his/ her color must be removed from the board, reducing his/ her mobility; and (2) the markers of the row must be removed, reducing the markers of his/ her color on the board.

---

[2] see www.gipf.com.

[3] See for example the Internet Top 100 Games List on http://scv.bu.edu/~aarondf/Top100/.

[4] In theory, a player could collect all three rings in one turn by making three rows by a single move. However, in practice this is an unlikely winning strategy.

5.3. Fair and balanced

The game-theoretic value of most games can easily be predicted without actually solving them: either the player with the initiative can force a victory, or the opponent has enough possibilities to force a draw. For example, at the highest level of competition, games of checkers and chess frequently result in a draw. Even though both games have not been solved yet, it is very likely that their game-theoretic value is a draw. An interesting aspect of dvonn, yinsh, and zèrtz is that their game-theoretic value is hard to predict.

Unlike chess and checkers, games between experienced players of dvonn rarely result in a draw.[5] After the placement phase, the board contains an odd number of pieces. So, even if a certain game progresses towards a close finish, an odd number of pieces should get isolated (and therefore must be removed) during the game to enable a draw. Dvonn is also very balanced because neither player has a clear advantage over the opponent: on the one hand, BLACK has the initiative in the placement phase, while WHITE has the initiative in the movement phase.

Balanced advantages and rare draws are also features of yinsh: although WHITE has the initiative in both the placement and the movement phases, this may not yield a clear advantage: after creating a row, a player has to remove a ring of his/ her color, which reduces his/ her mobility severely. A yinsh draw occurs only when all 51 markers have been placed on the board before any of the players has collected three of their rings.

The game of zèrtz never results in a draw: in the unlikely and extreme event that all board pieces would be filled with a marble (before neither of the players has a winning set of marbles), the player that made the last move, wins. In zèrtz, balanced advantages are guaranteed. There is no distinction between the material of the first and the second player. In general, the second player appears to have the upper hand in the opening phase of the game.[6] However, since the first player has numerous opening moves, there may still be an opening strategy assuring his/ her advantage.

The combination of balanced advantages in all three games and rare draws in both dvonn and yinsh— and no draws in zèrtz! — make it hard to predict their game-theoretic value.

5.4. Solvability

Solving the games of zèrtz, dvonn, and yinsh will be fertile grounds for the field of artificial intelligence: (1) It is a challenge to solve each of them; (2) All three games are very popular; (3) It is even hard to predict their game-theoretic value.

Clearly, zèrtz will be the easiest game to solve: it has only one starting position, and games between experienced players take only a few turns. Although zèrtz appears to offer the players a large number of possibilities per turn, in practice the possibilities are very limited: the player needs to avoid a deadly sequence of sacrifices by his/ her opponent. By developing pattern strategies based on these deadly sequences of sacrifices and on how to avoid them, the game may be solvable within a few years.

The most important obstruction to solve dvonn is its immense number of starting positions. Although not all starting positions should be evaluated to determine the game- theoretic value, those that need to be computed still amount to huge numbers. Besides this obstruction, it should be possible to solve a given starting position: (1) The game-tree is not very deep (at most 24 turns for each player); (2) transposition tables could be applied to reduce computational costs substantially; and (3) by threatening to isolate many pieces, a player could significantly tighten the number of moves of his/ her opponent.

For yinsh, none of the existing solving procedures seem to be applicable: (1) Retrograde analysis could not be applied because of its different nature; (2) flipping of the markings renders transposition tables useless; and (3) since three rings must be collected to win, none of the threat sequence procedures are profitable. Moreover, the billions of starting positions will make it even harder to solve the game. Therefore, yinsh will probably be a monumental challenge in this field for the next decades.

6. Conclusions and future work

The main conclusion of this paper is that the solvability of two-player games with perfect information could effectively be explained by the applicable solving procedures. The question which solving procedures could be

[5] See, for example, the championship results of dvonn on www.littlegolem.net.

[6] See for example http://www.scat.demon.co.uk/zertz/strategy.htm.

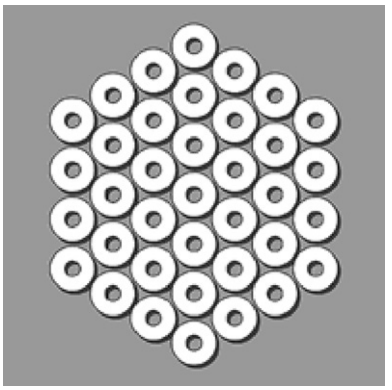

Fig. 8. The board at the start of the game.

applied to strongly solve a game is determined by the analysis of the characteristics of a game. We presented four such characteristics. Using these provides a more accurate classification than state-space and game-tree complexity would.

Five categories of solving techniques have been discussed. All of these techniques were mentioned in articles of currently solved games. However, only four appeared essential to obtain the known solutions. The only non-essential technique — game-tree search methods — is also the most diverse category: it contains many variants. The usefulness of these variants, in general or for a particular game, is unknown. Analyzing the performance of the techniques and comparing them to — for example — the minimal game-tree, will be a topic of future research.

In contrast to hand-made patterns, computer generated patterns are not yet an essential technique for strongly solving games. Still, we expect that solving hex on a large sized board, say of size 15×15, will require enormous patterns. Probably, their number will be too large to create by hand. By first solving hex on small sized boards — using only computer generated patterns analogous to the hand-made patterns by JingYang [21] — we will try to develop an effective solving procedure based on automatically generated patterns.

Appendix A.

See Table 4.

Appendix B. Zertz

Material: 6 white, 8 gray and 10 black marbles and 37 round board pieces.

Moving: The initial state of the board is shown in Fig. 8. During each turn, a player has two possible moves: (I) either place a marble on the board and then remove a board piece, or (II) capture one or more marbles.

(I) Placing a marble and removing a board piece:

1. At each turn, the player first selects a marble from the pool. Next, the marble must placed on the board. The player may select any color (s)he wishes, and (s)he may place the marble on any vacant board piece. *Important*: the marbles, in the pool as well as on the board, belong to both players (i.e. neither of the players "owns" the marbles placed on the board).
2. After a player has placed a marble on the board, (s)he must remove a "free" board piece. "Free" means: the piece must be vacant and it must be positioned at the edge of the board. In other words, there may not be a marble on it and the player must be able to remove it from the sides without disturbing the position of the remaining board pieces.

Table 4
Overview of procedures discussed in articles

Article	Retrograde analysis	Transposition tables	Game-tree search procedures	Winning threat-sequence procedures	Winning pattern procedures
Solving nine men's morris	DC, CP, NU	SY	$\alpha\beta$		
Building the checkers 10-piece…	**DC**, CP, NU				
Exploring the computational…	DC, **CP**				
Solving the game of awari…	DC, CP, **AA**	SY			
Replacement schemes for trans…		TI, **RS**, TS			
Solving 8×8 domineering		TI, **RS**, TS	$\alpha\beta$		HP
A knowledge-based approach…		SY, GP	CN		**HP**
The performance of PN*, PDS…			$\alpha\beta$, PN, **PN***, **PDS**		
Pentominoes: A first…		SY	HA		
Qubic: 4×4×4 tic-tac-toe		**SY**	HA	**Tseq**	
Search for solutions in games…			$\alpha\beta$, **PN**	**Tsp**	
Go-moku and threat-space…			PN	**Tsp**	
Solving renju			HA	Tseq	
Lambda search in game trees…				λ	
Domineering: Solving large…			$\alpha\beta$		**HP**
On a decomposition method…					**HP**
Search and evaluation in hex					HP, CP
Go patterns generated…		SY			**CP**
Partition search					**CP**

Explanation of abbreviations

Retrograde analysis:	DC: Decomposition, CP: Compression, AA: Asynchronic algorithm, NU: Neglecting unlikely positions.
Transposition tables:	TI: Table information, RS: Replacement schemes, TS: Table size SY: Symmetry, GP: Generalised positions.
Game-tree search procedures:	HA: by hand, $\alpha\beta$: α–β search , CN: Conspiracy numbers search, PN: Proof-number search.
Winning threat-sequence procedures:	Tseq: Threat-sequence search, Tsp: Threat-space search, λ: lambda search.
Winning patternprocedures:	HP: Hand-made pattern, CP: Computer generated pattern.

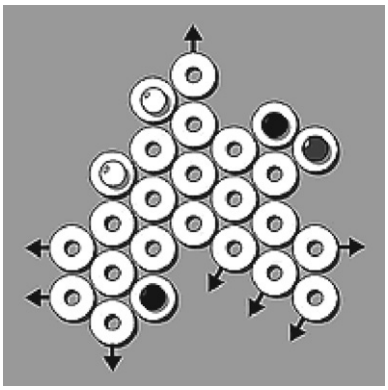

Fig. 9. Only the board pieces with an arrow may be removed.

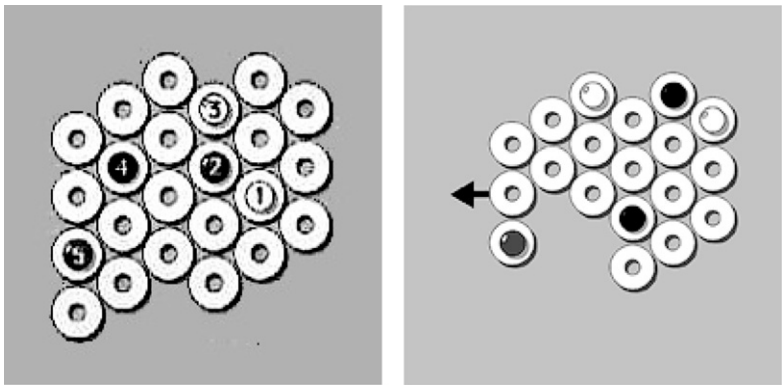

Fig. 10. (Left) Four possible captures: (a) 1 captures 2 and 3; (b) 1 captures 2, 4, and 5; (c) 2 captures 1; and (d) 3 captures 2 and 1. (Right) If you remove the board piece indicated by the arrow, you capture the marble on the isolated board piece.

3. Placing a marble and removing a board piece is one turn. A player must do both. However, it may occur that a player cannot remove any of the vacant board pieces without disturbing the position of the other board pieces. In this case, one does not remove a board piece (i.e. the move ends after having placed a marble).

(II) Capturing marbles:

1. If marbles could be captured, a player is forced to do so.
2. Marbles can be captured by jumping over them with another marble (i.e. as in checkers). One may only jump over a marble on an adjacent board piece. The player may jump in any direction if there is a vacant board piece behind the marble that (s)he intends to capture.
3. If the player jumps over a marble and (s)he has the possibility to jump over a second one, then (s)he must do so, no matter in which direction the second (or third) jump could be made.
4. When there are multiple ways to capture marbles (see Fig. 10 (left)), the player can freely chose one of them.
5. After capturing a marble, the player cannot not place a marble, nor remove a board piece.

Isolating marbles:

1. If a player succeeds in isolating one or more board pieces from the main part of the board, (s)he may claim these isolated pieces, including the marbles on them. Most of the time, it will concern one board piece, and thus one marble, but it is not limited to one. This “claiming” should be seen as a second way of capturing marbles, but it is not compulsory.
2. A player can only capture marbles this way if there are no vacant board pieces in the isolated group. So, a player may claim one or more board pieces when (s)he either puts a marble on the last vacant board piece of an already isolated group, or remove the board piece through which a group of occupied board pieces gets isolated.

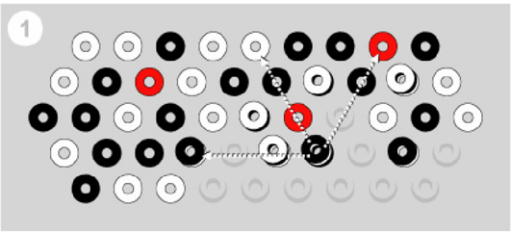

Fig. 11. The indicated stack may be moved 3 spaces in the direction of the arrows.

Goal: The goal of the game is to capture either 2 marbles of each color, or 3 white marbles, or 4 gray marbles, or 5 black marbles. The winner is the first player to achieve one of these goals.

Appendix C. Dvonn

Material: 23 white, 23 black and 3 dvonn pieces.

Placement phase: The game starts with an empty board. The players take turns placing their pieces on a vacant space of the board, one at a time. They must start with the dvonn pieces and then continue with their own color:

White places first dvonn piece; Black places second dvonn piece; White places third dvonn piece; Black places first black piece; White place first white piece; Black places second black piece; And so on. . . .

Moving phase:

1. White also begins the moving phase.
2. Each turn a player must move one piece or one stack. (S)he may only move a piece or a stack of his/ her own color. When two or more pieces are stacked on top of each other, the color of the topmost piece determines who owns the stack, and thus which player may move it.
3. A single piece may move one space in any direction, but only on top of another piece or stack of any color.
4. A stack must always be moved as a whole and moves as many spaces as there are pieces in the stack. Thus, a stack of 3 pieces (regardless of their color) must be moved exactly 3 spaces. Just like a single piece, a stack may be moved in any direction, but always in a straight line.
5. A move may never end in an empty space, but it is allowed to move across one or more empty spaces. When making a move, each space must be counted, no matter whether it is empty or occupied (see Fig. 11).
6. A piece or stack that is surrounded on all 6 sides may not be moved. So, at the beginning of the game, only the pieces at the edge of the board may move. The pieces that are not positioned at the edge remain blocked for as long as they remain completely surrounded (see Fig. 12).
7. A single dvonn piece may not be moved, but a piece or stack may move on top of it. When a dvonn piece is part of a stack, it is perfectly legal to move the stack containing the dvonn piece.
8. You may not pass a turn, unless you cannot make any more moves.

Losing pieces:

1. Pieces and stacks must somehow remain in contact with at least one dvonn piece to remain in play. “In contact” means that there must always be a link with at least one dvonn piece. Each and every piece and/or stack that is not linked to any of the dvonn pieces, must be removed from the board at once.
2. All removed pieces go out of the game. It doesn’t matter who makes the move through which the pieces and/or stacks become isolated. Watch out for this, especially in the endgame. Since you may not pass, you may be forced to make a move that isolates one or more of your own stacks.
3. All 3 dvonn pieces remain in play until the end of the game, even if one of them becomes isolated, as it will always remain in contact with itself.

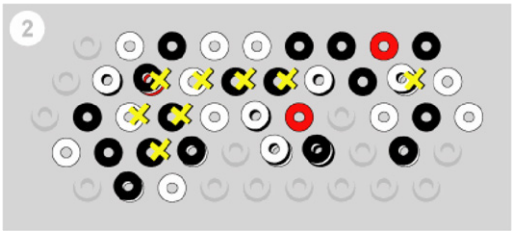

Fig. 12. The pieces and stacks marked with an X are completely surrounded.

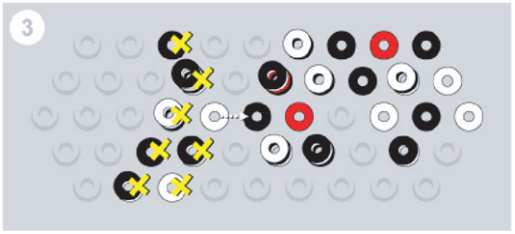

Fig. 13. If white moves the indicated piece, the pieces on the left will no longer be in contact with a dvonn piece. They must all be removed from the board at once.

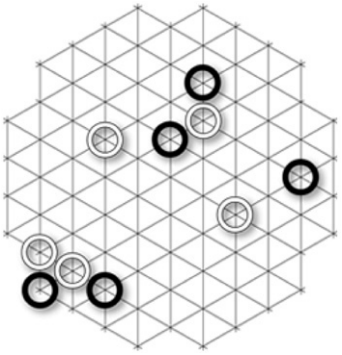

Fig. 14. The board after both players placed their rings.

Goal: The game ends when the last move is made. When this stage is reached, each player puts all of the stacks (s)he controls on top of each other. The player with the highest stack wins the game, regardless of the color of the pieces in his/ her stack. If both players end up with an equal stack, then the game ends in a tie.

Appendix D. Yinsh

Material: 5 white and 5 black rings, and 51 markers (white on one side, black on the other side).

Opening phase: White begins and places one of the white rings on the board, followed by Black placing one a black ring. This is repeated until both players have placed their five rings. After this opening phase the board looks like Fig. 14.

Moving:

1. Each move starts by a selection of one of the rings of the player.
2. Next, the player places a marker in that ring such that his color is facing up in the ring.
3. Then the player must move that ring according to the following rules:
   - Only the ring is moved, not the marker inside.
   - A ring must always move in a straight line and always to a vacant space.

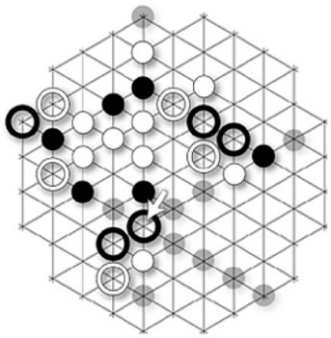

Fig. 15. A move. First you put a marker with your color face up in one of your rings, next you move the ring. You only move the ring, not the marker!.

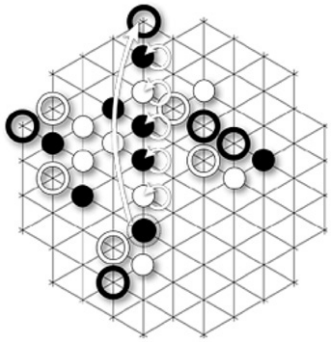

Fig. 16. The same situation as in Fig. 15, but now after black's move.

- A ring may move over one or more vacant spaces.
- A ring may jump over one or more markers, regardless of color, as long as they are lined up without interruption. In other words, if you jump over one or more markers, you must always put your ring in the first vacant space directly behind the markers you have jumped over.
- A ring can only jump over markers and vacant spaces, not over rings.

Collecting rings:

1. By moving rings and flipping markers, the players must try to form a row of 5 markers of their own color. These 5 markers must be adjacent and in a straight line. Rings do not count. For the sake of clarity: hereafter, a row of 5 markers that show the same color will simply be referred to as "a row".
2. After forming a row, the player must remove the 5 markers from the board.
3. After removing a row, the player must also collect one of the rings of this color. For this purpose, the player can choose any of his/ her rings.
4. If a player forms a row of more than 5 markers, then he can choose any 5 markers to remove — as long as they form an uninterrupted row.
5. It is possible to form two (or more) rows with only one move. If these rows don't intersect, the player must remove all these rows and as many rings. If rows do intersect, then the player may choose which row the opponent will take. Obviously, in this case, only one ring is collected.
6. A player could also form a row of the markers of the opponent. In this case, the opponent must remove the row and a ring before (s)he makes his/ her move. (S)he may freely choose which of his rings (s)he will collect.
7. If a player forms a row of his/ her own markers and also a row of his/ her opponent markers at the same time, then first (s)he removes his/ her own row (and a ring), and next turn the opponent removes his/ her row and a ring as described above.

Goal: The first player that collects three of his/ her own rings wins the game.

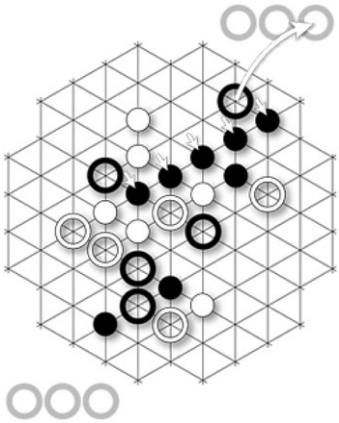

Fig. 17. Black has formed a row of 5 markers.

Acknowledgements

The first author was supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.306.

References

[1] L.V. Allis, Searching for solutions in games and artificial intelligence. Ph.D. Thesis, University of Limburg, The Netherlands, ISBN 90-9007488-0 (1994).
[2] L.V. Allis, H.J. van den Herik, M.P.H. Huntjens, Go-Moku and Threat-Space Search, http://citeseer.nj.nec.com/170657.html.
[3] L.V. Allis, H.J. van den Herik, I.S. Herschberg, Which games will survive? in: D.N.L. Levy, D.F. Beal (Eds.), Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad, Ellis Horwood, Chichester, ISBN: 0-13-382615-5, 1991, pp. 232–243.
[4] N. Bullock, Domineering: Solving large combinatorial search spaces, ICGA Journal 25 (2) (2002) 67–83.
[5] B. Bouzy, Go patterns generated by retrograde analysis. Computer Olympiad Workshop, 2001.
[6] D.M. Breuker, J.W.H.M. Uiterwijk, H.J. van den Herik, Replacement schemes for transposition tables, ICCA Journal 17 (4) (2004) 183–193.
[7] D.M. Breuker, J.W.H.M. Uiterwijk, H.J. van den Herik, Solving 8 × 8 domineering, Theoretical Computer Science (ISSN: 0304-3975) 230 (2000) 195–206.
[8] R.U. Gasser, Solving Nine Men's Morris, in: R.J. Nowakowski (Ed.), Games of No Change, vol. 29, MSRI Publications, 1996, pp. 101–113.
[9] M.L. Ginsberg, Partition Search, in: Procedings AAAI-96, 1996, pp. 228–233.
[10] H.J. van den Herik, J.W.H.M. Uiterwijk, J. van Rijswijck, Games solved: Now and in the future, Artificial Intelligence (ISSN: 0304-3975) 134 (1–2) (2002) 277–311.
[11] T.R. Lincke, Exploring the computational limits of large exhaustive search problems Ph.D. Thesis, ETH Zurich, Swiss, 2002.
[12] H.K. Orman, Pentominoes: A first player win, in: R.J. Nowakowski (Ed.), Games of No Change, vol. 29, MSRI Publications, 1996, pp. 339–344.
[13] O. Patashnik, Qubic: 4 × 4 × 4 Tic-Tac-Toe, Mathematics Magazine 53 (4) (1980) 202–216.
[14] J. van Rijswijck, Search and evaluation in Hex. Technical Report, University of Alberta, Canada, 2002.
[15] J.W. Romein, H.E. Bal, Solving the game of awari using parallel retrograde analysis, IEEE Computer 36 (10) (2003) 26–33.
[16] J. Schaeffer, Y. Bjornsson, N. Burch, R. Lake, P. Lu, S. Sutphen, Building the ckeckers 10-piece Endgame Databases, 2003.
[17] M. Sakuta, H. Iida, The Performance of PN*, PDS, PN Search on 6 × 6 Othello and Tsume-Shogi, in: H.J. van den Herik, B. Monien (Eds.), Advances in Computer Games, vol. 9, 2001, pp. 203–222.
[18] T. Thomsen, Lambda-search in game trees — With an application to Go, ICCA Journal 23 (4) (2000) 203–217.
[19] J.W.H.M. Uiterwijk, H.J. van den Herik, L.V. Allis, A knowledgebased approach to connect-four. The game is over: White to move wins! in: D.N.L. Levy, D.F. Beal (Eds.), Heuristic Programming in Artificial Intelligence: The First Computer Olympiad, 1989, pp. 113–133.
[20] J. Wágner, I. Virag, Solving Renju, ICGA Journal 24 (1) (2001) 30–34.
[21] J. Yang, S. Liao, M. Pawlak, On a decomposition method for finding winning strategy in Hex game, in: International Conference on Application and Development of Computer Games in the 21st Century, 2001, pp. 96–111.
[22] http://www.ee.umanitoba.ca/~jingyang/index.html.