# Quantified Boolean Solving for Achievement Games

Steve Boucher and Roger Villemaire

Dept. of Computer Science, UQAM, Montreal, Canada
steve.boucher@live.ca,villemaire.roger@uqam.ca

**Abstract.** Recent developments in the propositional representation of achievement games have renewed interest in applying the latest advances in Quantified Boolean Formula technologies to solving these games. However, the number of quantifier alternations necessary to explore the solution space still impairs and limits the applicability of these methods. In this paper, we show that one can encode blocking strategies for the second player and express the last moves of the play with a single string of existential quantifiers, instead of the usual alternations of universal and existential quantifiers. We experimentally show that our method improves the performance of state-of-the-art Quantified Boolean Formula solvers on Harary's Tic-Tac-Toe, a well-known achievement game.

**Keywords:** achievement game · Quantified Boolean Formula · winning strategy · Harary's Tic-Tac-Toe

## 1 Introduction

In an achievement game, also known as a positional game, two players take turns at adding a stone of their color on some board. The first player whose stones form a target *shape* wins the game. If no player achieves a target shape, the game is a *draw*. Tic-Tac-Toe, Gomoku, and Hex are the most well-known examples.

Typically, these games (in fact their generalizations to arbitrary board sizes) are PSPACE-complete. This is for instance the case for Gomoku [21] and Hex [20]. QBF, which ask whether a Quantified Boolean Formula is satisfied, or not, is the emblematic example of a PSPACE-complete problem [23]. It is therefore natural to apply QBF methods, developed in the wake of impressive advances in Propositional Satisfiability Testing (SAT) [1], to achievement games.

Games have also attracted the attention of theoreticians that have produced many involved combinatorial results [4]. In a way similar to the application of SAT solving to combinatorial problems, such as [14], it is hence also quite natural to apply QBF algorithms to the study of combinatorial games.

However, from a QBF perspective, the search for a winning strategy in a game gives rise to QBFs with many quantifier alternations, in stark contrast to most other application fields [22]. This is clearly a bottleneck, even if QBF game encodings have greatly improved recently [6, 18].

We present in this paper a QBF encoding for achievement games. We divert from the standard practice of encoding the existence of a winning strategy for the first player and rather search for a strategy for the second player that allows her to either win or get a draw. These two approaches are equivalent since the first player will win the game, if and only if, the second cannot win or get a draw. Moreover, we restrict the second player to follow, in the last steps of the game, a so-called pair paving strategy, which offers the benefit of being expressible with a string of existential quantifiers, instead of the usual alternation of universal and existential quantifiers.

With state-of-the-art QBF solvers, we evaluate our approach on Harary's Tic-Tac-Toe (HTTT), a game that has been extensively studied [12, 13, 3] and has also already been used to evaluate QBF game encodings [6, 18]. We show that our encoding leads to efficient QBF solving for HTTT, and furthermore that it can be combined with conventional winning strategy encodings to further improve solving time by iterative deepening.

This paper is structured as follows. Section 2 recalls basic notions on QBF, Section 3 presents Harary's Tic-Tac-Toe, and Section 4 recalls related work on QBF encodings of games. Section 5 presents our encoding and Section 6 our experimental evaluation. Finally, Section 7 concludes the paper.

## 2 Quantified Boolean Formulas

We consider *Boolean variables*, i.e., variables that can take value 1 ($true$) or 0 ($false$). A *literal* is either a variable or its negation. A *clause* is a disjunction (OR) of literals and a formula in *Conjunctive Normal Form* (CNF) is a conjunction (AND) of clauses. A CNF is furthermore *satisfiable* if there is an assignment of values to its variables that makes the formula true under the usual Boolean operators' semantics.

A *Quantified Boolean Formula (in prenex CNF form)* (QBF), is a CNF (the QBF's *matrix*) preceded by universal ($\forall$) and existential ($\exists$) quantifiers on the Boolean variables. QBF semantics can be recursively defined on the number of quantifiers. However, there is an alternative approach that is more relevant to this paper. Namely, one can consider a QBF to be a game between two players, *existential* and *universal*. The play follows the quantifiers from left to right; the existential player choosing a value for existentially quantified variables and the universal player choosing a value for universal quantified variables. The play ends when all quantifiers have been processed. The existential player then *wins* the game if the chosen assignment satisfies the QBF's matrix, otherwise the universal player *wins* the game. A QBF is then *satisfied* if the existential player has a winning strategy, namely if the existential player can make choices (that depend on the universal player's previous moves) assuring him to win the game.

## 3 Harary's Tic-Tac-Toe

F. Harary [7, 11] established the combinatorial analysis of achievement games by considering two players, usually called Black (first) and White (second), playing on an $N \times N$ square (regular) board with target shapes formed of the rotations and reflections of a *polyomino* (or *square animal*), which is an edge-connected set of cells. An animal for which the first player (Black) has a winning strategy is said to be a *winner*, otherwise it is a *loser*. A classic *strategy stealing* argument shows that there is in fact no winning strategy for White, since Black could play according to this strategy, by pretending that there is already some white stone on the board. So, when there is no winning strategy for Black, there is a *blocking* strategy for White that ensures a draw.

Fig. 1: Snaky

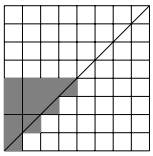

Fig. 2: Symmetry breaking

Harary has also established that 11 animals are winning when the board size is large enough and that all other animals, with a single exception, are losers. For these losers, he determined that White has a domino paving strategy, i.e., there is a partition of the board into 2-cells animals (dominoes), such that any target animal on the board contains at least one of these dominoes. White's blocking strategy is then simply to play on the second cell of the domino containing Black's previous move in order to block Black from winning the game.

For the unsettled case, that of *Snaky* (Fig. 1), Harary [7] conjectured that it is a winner, and furthermore that this should be the case for boards of size $15 \times 15$ and more, at least. This question, however, is still open.

As the last remaining case, it is not surprising that Snaky has attracted quite a lot of attention. For instance, Snaky has been shown to be a loser on $7 \times 7$ and $8 \times 8$ boards [10] by giving an explicit winning strategy for White that was found by a branch-and-cut algorithm that supplements a search for a blocking strategy for White, with a search for a paving of the board by pairs.

## 4 Related Work

Encodings of games have received some attention by the QBF research community. For instance, [9] gives a representation of the Connect-4 game in QBF. This encoding represents, with Boolean variables, the game configuration and move, in each turn, up to a maximum of $k$ turns. Since there is a maximum number of possible turns (for a fixed board size), $k$ is set to this value. A *gameover*

variable, one in each turn, is also used to determine the first player that wins the game. Finally, existential (universal) quantifiers represent moves of the first (second) player and the QBF expresses the existence of a winning strategy for the first player. However, this paper establishes that there are some concerns as to how to encode the rules of the game.

Indeed, adding game rules as conjuncts to the QBF's matrix would allow the second player to break a rule, falsifying the formula, and win the game. Similarly, adding game rules to the matrix as hypothesis in an implication would allow the first player to break a rule, satisfying the formula, and win the game.

The paper's solution is to introduce so-called *cheat variables*. These are existentially quantified variables for which conjuncts assuring their equivalence with a specific *cheat* (breaking of a rule) by a specific player are added to the matrix. Conjuncts are then added to the matrix so that a cheat by some player makes the other player win.

Following [9], a QBF encoding for HTTT and GTTT$(p, q)$, a generalization of HTTT allowing multiple stones to be laid in one move [5], is proposed in [6]. Experiments on 84 instances on a $3 \times 3$ board both normal and torus, and 96 instances on a $4 \times 4$ board, shows that all $3 \times 3$ instances can be solved in 10 seconds but that no combination of presented solvers and preprocessors allows to solve all $4 \times 4$ instances in 1000 seconds.

In [18] a general *corrective* encoding for achievement games is presented. The successive turns of the game are encoded, and a variable is used to determine when the game is over. However, contrary to [9] and [6], as soon as the game is over the board does not change anymore. This allows to check the winning condition solely in the last turn, strongly reducing the number of Boolean variables.

Another innovative aspect of this encoding is that universal variables represent White's (second player) choices in a binary encoding. Such a choice determines White's move at his turn, except if the game is already over, or if this choice is illegal (for instance if the position is already occupied). This allows a White's choice at every turn. The ladder encoding of [8] is furthermore used to ensure a single move of a player at her turn. This enforces a single move of White, even if his choice was illegal. All this prevents White from falsifying the formula, while keeping the total number of Boolean variables low.

The encoding is evaluated on 20 Hex hand-crafted puzzles and 96 instances of HTTT on a $4 \times 4$ board. It is shown that in terms of number of variables and clauses this encoding of HTTT is much more compact than that of [6] and solving time is much shorter. Furthermore, all 96 HTTT instances are solved within a timeout of 1000s. The paper finally proposes five game challenges for the QBF community including achieving Snaky on a $9 \times 9$ board.

## 5 The Pairing Encoding

For an achievement game, we consider blocking strategies for the second player, i.e., ensuring either a win by this second player or a draw. There is a winning strategy for the first player exactly when there is no blocking strategy for the

second player. Our QBF instances are hence satisfiable (SAT) exactly when the corresponding instance in one of the previous encodings is unsatisfiable (UNSAT). We furthermore restrict ourselves to the following pair paving blocking strategies.

A paving of the board by pairs, i.e., a partition of the board in pairs of distinct cells, such that any target shape contains at least one of those pairs, yields a so-called *pair paving blocking strategy* (for short a pair paving strategy) in the following way. The second player simply chooses the cell paired with the previous move of the first player. Since any target shape contains a pair, and hence a second player's stone, the first player does not win the game. Note that the existence of a pair paving simply necessitates a string of existential quantifiers, compared to an alternation of universal and existential quantifiers for a general strategy and should have a positive impact on QBF solving time.

While the existence of a pair paving strategy is enough to settle the game, its non-existence still leaves open the possibility that there could be some more involved blocking strategy for the second player. However, similarly to [10], it is possible to rather look for a pair paving strategy at *some* point of the game.

Our pairing encoding hence represents a play of the game where, at any point, the second player can stop the game and search for a pair paving of the remaining cells. Formally, the pairing encoding represents a game of length $k$, where the winning condition checks for a paving of the non-occupied cells by pairs of cells, such that any target shape, at any position, is *canceled*, i.e., contains either a second player's stone or a pair of the paving.

When $k$ is the maximal play length, our winning condition reduces to checking that any target shape, at any position contains a second player's cell, i.e., that the first player has lost the game. So, with maximal play length $k$, our instances will be UNSAT if and only if the first player has a winning strategy. It hence solves the status of the game but in a complementary way to the usual encodings.

For HTTT this approach should work well with the losers for which Harary has shown the existence of a domino (hence pair) paving strategy. However, Harary's winners could be losers on the board sizes of our experimental section, and there is no known pair paving strategy in these cases. Finally, Harary's considered only regular boards, while we also experiment on torus boards that "wraps around" on board edges.

To simplify the presentation, we will present the encoding in HTTT's setting. The Pairing encoding follows the corrective encoding of [18] but for the fact that Black/White are universal/existential players, since we are encoding a blocking strategy, and the winning condition is the existence of a paving.

The encoding is parameterized by $W$, $H$, the width and height of the board, and $k$ the number of turns. Time lapses $0, 1, \ldots, k$, the board being initialized at time 0 to contain no stones and moves occur at time points (turns) $1, \ldots, k$. A stone is added to the board at the same time point as the move occurs. Black hence plays at turns $1, 3, \ldots$, and White at turns $2, 4, \ldots$. There are $WH$ cells on the board, and we index pairs of cells by $id = 1, \ldots, WH(WH-1)/2$. Similarly, we index target shapes $i = 1, 2, \ldots$.

The variables representing the game, and their intended meaning, are defined as follows.

$$time_t;\ \text{the game is running at time point } t \tag{1}$$
$$moveL_{t,j};\ j\text{-th digit of the binary encoding of Black's move choices} \tag{2}$$
$$move_{t,x,y};\ \text{a stone is laid on cell } (x,y) \text{ at time point } t \tag{3}$$
$$ladder_{t,m};\ \text{ladder encoding for Black's moves (explained below)} \tag{4}$$
$$black_{t,x,y};\ \text{there is a black stone on cell } (x,y) \text{ at time point } t \tag{5}$$
$$white_{t,x,y};\ \text{there is a white stone on cell } (x,y) \text{ at time point } t \tag{6}$$
$$occupied_{t,x,y};\ \text{cell } (x,y) \text{ is occupied at time point } t \tag{7}$$
$$pair_{id};\ \text{the pair } id \text{ is in the paving} \tag{8}$$
$$canceled_{i,x,y};\ \text{shape } i \text{ at position } (x,y) \text{ is canceled} \tag{9}$$

Quantifier blocks appear in turn order as follows. In turn $t = 0$,

$$\exists time_0 \tag{10}$$

for $t = 1, \ldots, k$,

$$\exists time_t \tag{11}$$
$$\forall moveL_{t,j};\ \text{in Black's turns } t = 1, 3, \ldots, \tag{12}$$
$$\exists move_{t,x,y} \exists ladder_{t,m} \exists black_{t,x,y} \exists white_{t,x,y} \exists occupied_{t,x,y} \tag{13}$$

and as last quantifier block

$$\exists pair_{id} \exists canceled_{i,x,y} \tag{14}$$

The constraints are as follows, where $t = 1, \ldots, k$, $x = 1, \ldots, W$, and $y = 1, \ldots, H$.
If the game is still running at time $t$, it was running at time $t-1$.

$$time_{t-1} \vee \neg time_t \tag{15}$$

There is no stone on the board at time $t = 0$.

$$\neg black_{0,x,y} \wedge \neg white_{0,x,y} \wedge \neg occupied_{0,x,y} \tag{16}$$

Both players cannot own the same cell.

$$\neg black_{t,x,y} \vee \neg white_{t,x,y} \tag{17}$$

A stone on a cell remains there at the next turn.

$$(\neg black_{t-1,x,y} \vee black_{t,x,y}) \wedge (\neg white_{t-1,x,y} \vee white_{t,x,y}) \tag{18}$$

When the game is over, no new stones appear on the board.

$$(time_t \vee black_{t-1,x,y} \vee \neg black_{t,x,y}) \wedge (time_t \vee white_{t-1,x,y} \vee \neg white_{t,x,y}) \tag{19}$$

If a cell is played, then it is occupied.

$$(\neg black_{t,x,y} \vee occupied_{t,x,y}) \wedge (\neg white_{t,x,y} \vee occupied_{t,x,y}) \tag{20}$$

If a cell is occupied, then it is black or white.

$$\neg occupied_{t,x,y} \vee black_{t,x,y} \vee white_{t,x,y} \tag{21}$$

When the game is over, no more moves are allowed.

$$time_t \vee \neg move_{t,x,y} \tag{22}$$

The move is not allowed if the cell is already occupied.

$$\neg occupied_{t-1,x,y} \vee \neg move_{t,x,y} \tag{23}$$

A move sets a stone. Here *player* is either *black* when the turn $t$ is odd or *white* if it is even ($t > 0$).

$$\neg move_{t,x,y} \vee player_{t,x,y} \tag{24}$$

The universal player, Black, makes a move choice, specified by the $moveL_{t,j}$ bits assignation and the $move_{t,x,y}$ variables are set accordingly. More precisely, each cell choice $(x, y)$ is encoded by a string of bits $[x, y]$ of length $\lceil \log_2(WH) \rceil$. We will denote by $[(x, y)]_0$ the set of $i$'s for which the $i$-th bit in this string is 0 and by $[(x, y)]_1$ the set of $i$'s for which it is 1.

We therefore have, for Black's turns, $t = 1, 3, \ldots$, that if the game is not over and the cell $(x, y)$ not occupied at the previous turn, a choice of $(x, y)$ (by the $moveL$ variables) performs a move on $(x, y)$ ($move$ variables).

$$\neg time_t \vee occupied_{t-1,x,y} \vee \bigvee_{j \in [x,y]_0} moveL_{t,j} \vee \bigvee_{j \in [x,y]_1} \neg moveL_{t,j} \vee move_{t,x,y} \tag{25}$$

By symmetry under rotations, it is sufficient to consider the case where the first player (Black) plays on the lower left quarter in her first turn, and by reflection (on the diagonal, see Fig. 2) this can be further reduced to the lower left triangle of the square board. When generating clauses (25), for the first move of the first player, only choices of moves in this triangle is hence considered.

For $player, other \in \{white, black\}$, *player* being the color of turn $t$'s player and *other* the color of the other player, we have that a player's stone that was not placed at this turn, was already there at the previous turn. Similarly, the other player's stones were already there at the previous turn.

$$(move_{t,x,y} \vee player_{t-1,x,y} \vee \neg player_{t,x,y}) \wedge (other_{t-1,x,y} \vee \neg other_{t,x,y}) \tag{26}$$

In order to ensure that, at each turn where the game is still running, a single move is done, the corrective encoding [18] uses the ladder encoding of [8]. There

are hence $ladder_{t,m}$ variables, one for each possible move $m$. To ensure that a single move is done, the move $m$ is determined from the position of the "shift", from 0 to 1, in the ladder variables. Here, to ease the presentation, the moves (or equivalently the ladder variables for a turn) are considered ordered, in some arbitrary way. The key point is that to ensure a unique shift, the last ladder variable is set to true, and the ladder variables are constrained to be increasing.

More precisely, we have the following constraints.

Ladder variables are increasing and when the game is running, the last ladder variable is true.

$$(\neg ladder_{t,m} \vee ladder_{t,m+1}) \wedge (\neg time_t \vee ladder_{t,last}) \tag{27}$$

When the "shift" in the ladder variables occurs, this determines the move. Therefore, for the first move $m = first$ in the ladder encoding order, if the variable $ladder_{t,first} = 1$, then move $m$ is done. For a move $m$ different from the first, if $ladder_{t,m-1} = 0$ and $ladder_{t,m} = 1$, then move $m$ is done. Here $(x, y)$ is the destination of move $m$.

$$(\neg ladder_{t,first} \vee move_{t,x,y}) \wedge (ladder_{t,m-1} \vee \neg ladder_{t,m} \vee move_{t,x,y}) \tag{28}$$

Conversely, if a move is done, the "shift" in the ladder encoding occurs at that move. Therefore, if the move $m$ to $(x, y)$ is done, then $ladder_{t,m} = 1$ and, but for the first move, $ladder_{t,m-1} = 0$.

$$(\neg move_{t,x,y} \vee ladder_{t,m}) \wedge (\neg move_{t,x,y} \vee \neg ladder_{t,m-1}) \tag{29}$$

We finally add our winning pair paving condition.

We first have to ensure that if a pair of cells is in the paving, then no cell of this pair is occupied. Hence, for a pair of cells $id$ and $(x, y)$ one of its cells, we have the following constraint.

$$\neg pair_{id} \vee \neg occupied_{k,x,y} \tag{30}$$

Furthermore, no two pairs of the paving have a common cell. Therefore, for distinct pairs $id1, id2$ with a common cell we have the following clause.

$$\neg pair_{id1} \vee \neg pair_{id2} \tag{31}$$

Also, if a pair is in the paving, it cancels any shape containing it. We therefore have, for all pairs $id$ contained in shape $i$ at position $(x, y)$, the following clause.

$$\neg pair_{id} \vee canceled_{i,x,y} \tag{32}$$

Furthermore, if the cell contains a white stone, then a target shape at some position that contains this cell is canceled. We hence have, for all cells $(x', y')$ contained in shape $i$ at position $(x, y)$, the following clause.

$$\neg white_{k,x',y'} \vee canceled_{i,x,y} \tag{33}$$

Conversely, if a target shape at some position is canceled, then either one of its cells contains a white stone or it contains some pair. We therefore have, for target shape $i$ at position $(x, y)$, $\mathcal{C}$ its set of cells and $\mathcal{P}$ the set of pairs contained in this shape at this position, the following clause.

$$\neg canceled_{i,x,y} \vee \bigvee_{(x',y')\in\mathcal{C}} white_{k,x',y'} \vee \bigvee_{id\in\mathcal{P}} pair_{id} \tag{34}$$

Finally, any shape $i$ at any position $(x, y)$, is canceled.

$$canceled_{i,x,y} \tag{35}$$

## 6 Experimental Results

All experiments are run on a 2 CPU X5570 Xeon Processor, 2.93GHz, 64GB Memory, with a timeout of 1000 seconds. We consider HTTT instances for polyominoes formed by at most 6 cells that fit on the board, and this both for normal and torus boards. We hence have 48 instances on $3 \times 3$, 98 instances on $4 \times 4$, and 110 instances on $5 \times 5$ boards. We also show some results about the Snaky polyomino. We used the following QBF solvers versions: DepQBF v6.03 [16], CAQE v4.0.1 [24], Qute v1.1 [19], QESTO v1.0 [15], and four preprocessors (including none): QRATPre+ v2.0 [17], HQSPre v1.4 [26], and bloqqer v37 [2]. Every system is used with default command line parameters.

Table 1: $3 \times 3$ solving time in seconds with and without preprocessor

| Solver | Preprocessor | DYS | COR | Pairing |
|---|---|---|---|---|
| DepQBF | None | **12.85** | **1.12** | **0.77** |
| QESTO | None | 3171.66 | 9.21 | 3.00 |
| DepQBF | QRATPre+ | 7.97 | **0.85** | **0.32** |
| QESTO | bloqqer | **3.47** | 1.65 | 1.69 |

On $3 \times 3$ boards we compare the DYS encoding of [6], the corrective encoding COR [18] and our own Pairing encoding. We show in Table 1 the best solver, with no processor, and the best pair (solver/preprocessor) on $3 \times 3$ boards. There was no timeout in this case and there are exactly 8 winners and 40 losers. In all cases, both with and without preprocessor, the Pairing encoding allows the shortest solving time. Furthermore, confirming the results of [18], the DYS encoding is largely less efficient. For boards of size $4 \times 4$ and more, we hence concentrate on the COR and Pairing encodings.

On a $4 \times 4$ board (Table 2), a striking observation is that, in all cases, the Pairing encoding is more than two orders of magnitude faster than the COR encoding, with no timeouts. Preprocessing time (Table 3) is also much shorter for bloqqer and QRATPre+ than for HQSPre, but still, it is, for these two preprocessors, comparable to solving time for the Pairing encoding.

Table 2: $4 \times 4$ solving time in seconds, numbers of Unknowns, Winners, Losers

| Solver | Preprocessor | COR | | | | Pairing | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | time | U | W | L | time | U | W | L |
| DepQBF | None | **17159.72** | **7** | 14 | 77 | **114.63** | **0** | 14 | 84 |
| DepQBF | bloqqer | **12949.31** | **5** | 14 | 79 | 93.35 | 0 | 14 | 84 |
| DepQBF | QRATPre+ | 14176.01 | 5 | 14 | 79 | **61.31** | **0** | 14 | 84 |

It is also interesting to compare COR and Pairing in terms of instances size. We show in Table 4 total number of literals, clauses, and quantifiers over all shapes for normal and torus boards. While numbers of quantifiers are similar for both encodings, Pairing generates more literals and clauses, in particular for torus boards.

Table 3: $4 \times 4$ preprocessing time in seconds

| | COR | | | Pairing | | |
|---|---|---|---|---|---|---|
| | bloqqer | HQSPre | QRATPre+ | bloqqer | HQSPre | QRATPre+ |
| avg. | 1.78 | 27.98 | 0.10 | 1.87 | 8.99 | 0.38 |
| max. | 2.11 | 65.39 | 0.20 | 2.25 | 40.95 | 0.71 |
| total | 174.03 | 2742.51 | 9.65 | 183.30 | 881.19 | 37.31 |

On $5 \times 5$ boards we tested only the solver and solver/preprocessors pairs that gave the best performance on $4 \times 4$ boards (as shown in Table 2). We see, in Table 5, that COR solves only 7 out of 110 instances, while Pairing solves 72 (70 with no preprocessor). As for run time, Pairing needs less than half the time required by COR. Pairing hence again clearly outperforms COR.

Table 4: Total number of literals, clauses, and quantifiers on $4 \times 4$ boards

| | COR | | | | Pairing | | | |
|---|---|---|---|---|---|---|---|---|
| | lits | clauses | Univ. | Exist. | lits. | clauses | Univ. | Exist. |
| normal | 554629 | 227944 | 1568 | 66850 | 675390 | 285428 | 1568 | 70682 |
| torus | 637683 | 253794 | 1568 | 70241 | 950037 | 388083 | 1568 | 75329 |

Run time increases rapidly with play length $k$ and solving the game necessitates a $k = 25$ on a $5 \times 5$ board, while $k = 9$ and $k = 16$ suffice for $3 \times 3$ and $4 \times 4$ boards, respectively. It could hence be appropriate to apply iterative deepening that could establish a winning strategy for a smaller value of $k$. However, we go a step further and combine the corrective encoding of [18] that searches for a winning strategy for Black and our encoding that search for a blocking strategy for White. This allows to establish, as soon as either return a SAT instance, the status (winner/loser) of the polyomino.

Table 6 shows the cumulative time of solving instances for increasing values $k = 0, 1, 2, \ldots, 25$ using the Pairing encoding for even $k$ and the COR encoding for odd $k$, stopping at the first SAT instance.

Table 5: $5 \times 5$ solving time in seconds, numbers of Unknowns, Winners, Losers

| Solver | Preprocessor | COR | | | | Pairing | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | time | U | W | L | time | U | W | L |
| DepQBF | None | **103083.14** | **103** | 7 | 0 | 41365.22 | 40 | 10 | 60 |
| DepQBF | bloqqer | 103213.42 | 103 | 7 | 0 | **38901.20** | **38** | 10 | 62 |
| DepQBF | QRATPre+ | 103114.56 | 103 | 7 | 0 | 39393.55 | 38 | 10 | 62 |

Comparing Tables 5 and 6, one sees that iterative deepening combining COR and Pairing is indeed effective. It, in fact, reduces the total number of timeouts and solving time almost by one half, compared to Pairing that gives the best results in Table 5.

Table 6: $5 \times 5$ iterative deepening solving time in seconds, numbers of Unknowns, Winners, Losers

| Solver | Preprocessor | normal | | | | torus | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | cum. time | U | W | L | cum. time | U | W | L |
| DepQBF | None | 1023.71 | 1 | 7 | 47 | 21376.28 | 21 | 7 | 27 |
| DepQBF | bloqqer | 1016.29 | 1 | 7 | 47 | 21012.67 | 20 | 7 | 28 |
| DepQBF | QRATPre+ | 1015.34 | 1 | 7 | 47 | 21310.61 | 20 | 7 | 28 |

Table 6 further shows that there is a stark difference between normal and torus boards. Another surprising fact on $5 \times 5$ instances is that preprocessors yield very small performance gain, both in terms of solving time than in terms of number of solved shapes. This is in stark contrast to $4 \times 4$ and $3 \times 3$ boards.

It is instructive to look at the distribution of the last considered $k$ with this iterative deepening method on $5 \times 5$ boards. Fig. 3 shows, for each $k = 0, \ldots, 11$, the number of polyominoes for which the first SAT instance (or timeout) is reached for this value of $k$. Note that there are polyominoes settled at each intermediate value of $k$, both even and odd. It follows that neither even nor odd $k$'s can be avoided and both COR and Pairing are instrumental in making this method effective.

Obviously, the big open question around HTTT is the status of Snaky on a $9 \times 9$ board, and as stated by [18] this is a challenge to the QBF community. It is hence interesting to see how the Pairing encoding fares with this polyomino. However, as shown in Table 7 that shows solving time in seconds for DepQBF on normal boards, the loser status of Snaky on an $8 \times 8$ board [10] is already out of reach of our Pairing encoding. Solving Snaky on a $9 \times 9$ board will hence need further advances in QBF solving and encoding.

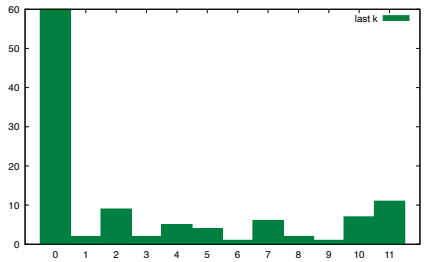

Fig. 3: last k, DepQBF

Table 7: Snaky Pairing

| board | k | result | time |
|---|---|---|---|
| $6 \times 6$ | 0 | UNSAT | 1.53 |
| | 2 | SAT | 17.53 |
| $7 \times 7$ | 0 | UNSAT | 0.81 |
| | 2 | UNSAT | 345.64 |
| | 4 | SAT | 2277.39 |
| $8 \times 8$ | 0 | UNSAT | 0.79 |
| | 2 | UNSAT | 1857.64 |
| | 4 | UNSAT | 10596.75 |
| | 6 | UNSAT | 120004.47 |

## 7 Conclusion

We showed that the QBF Pairing encoding leads to very effective QBF solving for HTTT on $3 \times 3$ and $4 \times 4$ boards. Furthermore, iterative deepening combining the COR and Pairing encodings can solve HTTT on $5 \times 5$ boards for most target shapes. However, determining shapes on a $5 \times 5$ torus board often necessitates higher values of $k$, and would require further development in QBF game solving.

On a methodological level our contribution is to draw attention to the effectiveness, in the QBF solving setting of achievement games, of considering "dual" encodings with blocking strategies, and furthermore restricting the kind of blocking strategy sought.

Indeed, while describing in QBF the existence of a winning strategy for the first player is the natural approach, one can equivalently encode the existence of a blocking strategy by the second player. This is reminiscent of the use of primal and dual encodings [25] that have been shown to both have their advantages in the QBF analysis of synchronous system.

Furthermore, in order to show the existence of a winning strategy, one can as well show the existence of some specific kind of strategy, as this could lead to further performance gain, as is the case with the Pairing encoding for HTTT.

While our results advance the QBF solving of achievement games, it still come short of solving Snaky's status on a $9 \times 9$ board. This will need further advances in QBF solving, where "dual" encodings and restricting the kind of sought strategy could play a role.

## References

1. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
2. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Automated Deduction - CADE-23. LNCS, vol. 6803, pp. 101–115. Springer (2011)
3. Csernenszky, A., Martin, R.R., Pluhár, A.: On the complexity of chooser-picker positional games. INTEGERS: Electronic Journal of Combinatorial Number Theory **11**(G2) (2011)

4. Demaine, E.D., Hearn, R.A.: Playing games with algorithms: Algorithmic combinatorial game theory, p. 3–56. Mathematical Sciences Research Institute Publications, Cambridge University Press (2009)
5. Diptarama, Narisawa, K., Shinohara, A.: Drawing strategies for generalized tictactoe $(p, q)$. AIP Conference Proceedings **1705**(1), 020021 (2016)
6. Diptarama, Yoshinaka, R., Shinohara, A.: QBF encoding of generalized Tic-Tac-Toe. In: Quantified Boolean Formulas, QBF 2016. CEUR Workshop Proceedings, vol. 1719, pp. 14–26 (2016)
7. Gardner, M.: Mathematical games. Scientific American **240**(4), 18–28 (April 1979)
8. Gent, I., Nightingale, P.: A new encoding of all different into SAT. In: Modelling and Reformulating Constraint Satisfaction Problems. pp. 95–110 (2004)
9. Gent, I., Rowley, A.R.: Encoding Connect-4 using quantified Boolean formulae. In: Modelling and Reformulating Constraint Satisfaction Problems. pp. 78–93 (2003)
10. Halupczok, I., Schlage-Puchta, J.C.: Achieving Snaky. INTEGERS: Electronic Journal of Combinatorial Number Theory **7**(G02) (2007)
11. Harary, F.: Achieving the Skinny animal. Eureka **42**, 8–14 (1982)
12. Harborth, H., Seemann, M.: Snaky is an edge-to-edge loser. Geombinatorics **V**(4), 132–136 (1996)
13. Harborth, H., Seemann, M.: Snaky is a paving winner. Bulletin of the Institute of Combinatorics and its Applications **19**, 71–78 (1997)
14. Heule, M.J.H., Szeider, S.: A SAT approach to clique-width. ACM Transactions on Computational Logic **16**(3), 24 (2015)
15. Janota, M., Marques-Silva, J.: Solving QBF by clause selection. In: International Joint Conference on Artificial Intelligence, IJCAI. pp. 325–331. AAAI Press (2015)
16. Lonsing, F., Egly, U.: DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. In: Automated Deduction - CADE 26. LNCS, vol. 10395, pp. 371–384. Springer (2017)
17. Lonsing, F., Egly, U.: QRATPre+: Effective QBF preprocessing via strong redundancy properties. In: Theory and Applications of Satisfiability Testing - SAT. LNCS, vol. 11628, pp. 203–210. Springer (2019)
18. Mayer-Eichberger, V., Saffidine, A.: Positional games and QBF: The corrective encoding. In: Theory and Applications of Satisfiability Testing, SAT 2020. LNCS, vol. 12178, pp. 447–463. Springer (2020)
19. Peitl, T., Slivovsky, F., Szeider, S.: Qute in the QBF evaluation 2018. Journal on Satisfiability, Boolean Modeling and Computation **11**(1), 261–272 (2019)
20. Reisch, S.: Hex ist PSPACE-vollständig (Hex is PSPACE-complete). Acta Informatica **15**, 167–191 (1981)
21. Reisch, S.: Gobang ist PSPACE-vollständig (Gomoku is PSPACE-complete). Acta Informatica **13**, 59–66 (1980)
22. Shukla, A., Biere, A., Pulina, L., Seidl, M.: A survey on applications of Quantified Boolean Formulas. In: International Conference on Tools with Artificial Intelligence, ICTAI 2019. pp. 78–84. IEEE (2019)
23. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time: Preliminary report. In: ACM Symposium on Theory of Computing (STOC). pp. 1–9. ACM (1973)
24. Tentrup, L.: CAQE and QuAbS: Abstraction based QBF solvers. Journal on Satisfiability, Boolean Modeling and Computation **11**(1), 155–210 (2019)
25. Van Gelder, A.: Primal and dual encoding from applications into quantified Boolean formulas. In: Principles and Practice of Constraint Programming, CP 2013. LNCS, vol. 8124, pp. 694–707. Springer (2013)

26. Wimmer, R., Reimer, S., Marin, P., Becker, B.: HQSpre - an effective preprocessor for QBF and DQBF. In: Tools and Algorithms for the Construction and Analysis of Systems TACAS. LNCS, vol. 10205, pp. 373–390 (2017)