Using Games for Benchmarking and Representing the Complete Solution Space using Symbolic Techniques

Görschwin Fey Sebastian Kinder Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{fey,kinder,drechsle}@informatik.uni-bremen.de

Abstract

Games often are inherently multi-valued problems and their wide variety offers different graduations of complexity. Moreover a lot of games have a parameter, like board-size that allows to generate differently sized instances of the same problem. All this makes them perfectly suitable for benchmarking in the multi-valued domain. So far the lack of benchmarks in this area often was compensated by transferring problems from binary to multi-valued, but for several application domains this is not adequate.

This paper focuses on three games, that we consider suitable for benchmarking. We show the differences in complexity of the games and compare two coding schemes for one of them. All three problems are modeled by symbolic techniques, namely decision diagrams, leading to a complete representation of the solution space. This representation finds several applications, e.g. in objectively analyzing the efficiency of different heuristics on a solution space or to speed up learning algorithms.

1. Introduction

Since most algorithms in the area of computer aided design are based on heuristics, their quality is usually evaluated using benchmark sets [3]. But so far when considering multi-valued circuits often binary benchmark circuits were taken and transfered into multi-valued circuits to provide results for certain techniques (see e.g. [14, 9]), but no specific test cases from the multi-valued domain were used.

Now, looking for benchmarks in the multi-valued domain, games immediately come into focus for three reasons:

1. Most games are inherently multi-valued, e.g. they often have a board of several rows and columns.
2. The encoding of a game is not determined, but only rules are given. Therefore different coding styles can be compared.
3. Games are usually scalable, so the complexity of a problem can be increased. This is useful to analyze the scalability of a particular solution strategy.

In this paper we focus on three games: the n Queens problem, Tic-Tac-Toe and Connect Four. The n Queens problem, is a combinatorial problem, but easily scalable. Expanding the solution space by increasing n soon leads to complexity problems. Tic-Tac-Toe is a game that has a notion of time and by this it requires for a sequential model. We compare two encoding schemes to model this game. Finally, Connect Four is a sequential problem, too. But this game is more complex and therefore modeling and representing the whole game in a decision diagram is difficult. Besides the maximum size of the board, the number of moves that can be modeled provides information about the scalability of the applied technique.

We apply binary decision diagrams [4] as a symbolic technique to model the games. The use of decision diagrams has been studied earlier on the "Number of Knight Tours Problem" [8, 7] and also on combinatorial problems in general [10, 5]. In both cases the aim was to solve the particular problem. Here, the suggestion is to exploit the symbolic representation that speeds up the evaluation of candidate solutions. This can be of use in objectively analyzing the efficiency of heuristics on the solution space or to speed-up learning. A similar concept has been used for

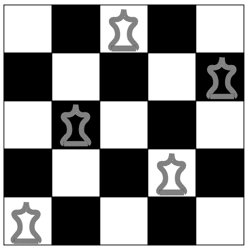

Figure 1. One solution of the 5 Queens problem

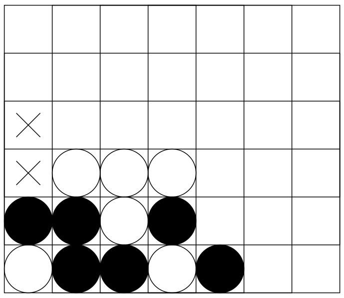

Figure 2. Connect Four: White will win

fast simulation of integrated circuits: Instead of slowly calculating the output values by traversing the circuit, they are directly evaluated from the input values using the symbolic representation [2]. The same applies to a candidate solution for a problem. The time consuming calculations to evaluate the validity can be replaced by a fast evaluation with the help of the decision diagram. From this speed up for example the learning of heuristics [6] can benefit. The heuristic can be learned on a small instance of a problem and then be applied to the larger instances that can not be fully modeled any more.

The paper is structured as follows: In Section 2 the three games are briefly introduced, their rules are explained and properties are discussed. Section 3 shows a scheme how to model sequential games by using symbolic techniques and details about the models for our particular problems. Tic-Tac-Toe will be modeled by two different encoding schemes. Experimental results are given in Section 4, where scaling of the n Queens problem is investigated, the two approaches to code Tic-Tac-Toe are compared and the difficulties at modeling Connect Four are shown. Conclusions and a discussion close the paper.

2. The Games

n Queens The problem is to place n queens on a $n \times n$ chess board in a way that no queen can be captured by any other queen.

A solution to the 5 Queens problem is shown in Figure 1. This is a combinatorial problem which is not difficult to solve for small instances, but it is scalable. The number of elements in the space of possible placements of n queens on the board obviously is $\binom{n^2}{n}$ - most of them are not a solution to the problem.

Tic-Tac-Toe Tic-Tac-Toe is a well-known game played by two players on a 3×3 grid. Each player in turn places a stone on the field, one player places circles, the other one 'x'-es. The player who places circles begins. The aim is to place three stones side by side.

This is a sequential problem, since the order of moves is important. A trivial upper bound to the number of valid sequences of moves is 9!, since each field of the board can only be occupied once. So the solution space is not too large.

If both players always choose their best move, the game ends in a draw.

Connect Four The board of Connect Four is 6 rows times 7 columns wide and standing upright. Therefore a stone put into one column falls down to the bottom or the topmost stone in that column. Two players having white and black stones are turn-wise trying to connect four of their own stones in a vertical, horizontal or diagonal line. White begins.

Example 1. *Look at Figure 2 to see a forlorn situation of black. White can complete a row of four stones by placing a stone on either of the fields that are marked by '$\times$'. By this - even if it is blacks turn - white can win the game by throwing the next stone into the first column.*

There are 42 fields on the board, each of which can be in one of three states: white, black or blank. This would lead to a large number of possible situations, that obviously can be heavily pruned. By applying restrictions only those situations have to be handled that arise from valid sequences of moves.

This game has been solved by knowledge-based techniques in [1]: White will always win, if the player always chooses the best move. But here we consider the problem of representing all solutions in parallel.

3. Modeling the Games

To be suitable for a comparison to other approaches and so serve as a reference point details about the used modeling technique are explained in this section. At first a general concept to model sequential games is introduced. Afterwards the coding of each problem is presented in more detail.

3.1. Sequential Modeling Algorithm

The correspondence of sets and their representation by a symbolic technique was discussed in [10] already. In that case zero-suppressed binary decision diagrams were used as the data structure for symbolically representing a set. All of the results on zero-suppressed binary decision diagrams can directly be mapped to binary decision diagrams, that are used in this work. In the following we assume the reader is familiar with binary decision diagrams, more details can be found in [4].

The n Queens problem is combinatorial and can be viewed as a game that has only one move. Therefore all games mentioned can be modeled by coding each move in one or more input variables. By applying constraints on these variables it is possible to reduce the explored space to valid sequences of moves. In a next step the conditions for a victory of one player are applied, thereby retrieving all valid sequences of moves that lead to a victory of one player.

The ith move m^i is encoded by s binary input variables: $m^i = m^i_1 \dots m^i_s$. The constraints are built after each move and denoted by $c^j(m^0, \dots, m^j)$, so are the winning conditions $v^j(m^0, \dots, m^j)$.

When modeling a 'real' sequential game, which is the case for Tic-Tac-Toe and Connect Four, those sequences that led to a victory of one player are collected and must not be extended by further moves. Therefore the function of winning assignments for the games can be calculated by the induction given in Figure 3. It calculates an expression win that represents all valid sequences of moves that lead to a victory of either player. win^j represents all sequences of up to j moves that lead to a victory of one player, all^j represents all valid sequences of up to j moves that did not lead to a victory. During the step of the algorithm (lines 3-10) at first the valid sequences that lead to a victory in the jth move are calculated in tmp. These are added to the winning sequences of the previous step (line 5) and the remaining valid sequences are calculated (line 6). Finally the next step is carried out, if there are more moves to make.

Initialization
(0) $win^0 \leftarrow 0$
(1) $all^0 \leftarrow 1$
(2) $j \leftarrow 1$
Step
(3) Calculate v^j and c^j
(4) $tmp = all^{j-1} \wedge v^j \wedge c^j$
(5) $win^j = win^{j-1} \vee tmp$
(6) $all^j = all^{j-1} \wedge \overline{tmp}$
($= all^{j-1} \oplus tmp$, since $tmp \leq all^{j-1}$)
(7) IF ($j <$ maximum number of steps) THEN
(8) $j \leftarrow j + 1$
(9) GOTO (3)
(10) FI
Result
(11) OUTPUT win^j

Figure 3. Modeling a sequential game

Of course the algorithm can easily be modified to output all valid runs instead of only those that lead to a victory. This is done by changing line 11 to "OUTPUT $win^j \vee all^j$".

This algorithm maps the sequential problem to a combinatorial one. By using symbolic methods to represent the constraints and conditions all possible winning sequences are given in parallel.

Note the analogy of the algorithm to reachability analysis for automatons as applied in formal verification.

3.2. n Queens

This problem is essentially non-sequential. Given a placement of queens on the chess board the decision if it is valid has to be made. Therefore it can be viewed as having only one move.

This move has n^2 binary input variables, each one deciding, if a queen is placed on the corresponding field of the chess board or not. Obviously the constraints are to place at most one queen per row, column and diagonal. The winning condition is to place at least n queens, which can be simplified to e.g. placing at least one queen per row.

This problem is suitable to be modeled in a multi-valued representation, due to its notion of rows and columns.

3.3. Two Approaches to Tic-Tac-Toe

Tic-Tac-Toe is modeled sequentially. Each move designates the field where the player places the stone, odd moves correspond to circles even moves to 'x'-es. Of course a game ends after at most 9 moves. Tic-Tac-Toe was implemented using two ways to code a move. This shows the influence of the coding scheme on the difficulty of solving the problem.

One-Hot Coding Scheme Each move consists of 9 input variables corresponding to the 9 fields of the board. One of the input variables is set in each move, so the field of this move can easily be determined. Obviously 81 input variables are needed to code all nine moves of one game.

Binary Coding Scheme In this model each move was coded by only four binary variables, two of them giving the column, the other two the row of the move. This leads to only 36 variables to code the game.

The constraints are the same for both coding schemes, even though the particular expressions are different. First of all each move must point to an existing field on the board. This is trivially true in the one-hot coding scheme. In the other case the values for the inputs are constraint. This first constraint is not complex, since it only takes the variables related to the current move into account. The second constraint assures that only one stone is placed on each field, i.e. no two moves have the same input value. Therefore this is more complex than the previous constraint. This time all previous moves are involved as well, so the logical expression to model the constraint includes all variables of previous moves and those of the current move.

The winning condition is obvious: One player wins, if the current move finishes a row, column or diagonal of three of his stones. Here again, all variables of all moves up to the current one have to be evaluated in the logical expression. Furthermore the condition for a player to win is independent of the order the stones were placed next to each other. Therefore all permutations have to be considered as well.

Example 2. *Consider the binary coding scheme. The column of the* j*th move is coded by variables* m_0^j *and* m_1^j*. The row is coded by* m_2^j *and* m_3^j*. For this move there are two constraints.*

1. *Row and column are within the range* $0 \ldots 2$*, i.e.*

$$\overline{m_0^j m_1^j} \wedge \overline{m_2^j m_3^j}$$

2. *The move is different from all previous moves, i.e.*

$$\bigwedge_{i=0}^{j-1} \bigvee_{k=0}^{3} m_k^i \neq m_k^j$$

3.4. Connect Four

This game has a notion of time, too. Each move is coded by three input variables, selecting between the seven columns the one the stone is put into. As previously mentioned the color of a stone is different for even and odd moves. The game ends after at most 42 moves.

Again, a first constraint is to restrict each move to one of the seven columns of the board. Somewhat more complex is the second constraint, the restriction to six stones per column. This constraint requires to include all previous moves in the expression. Even more complex becomes the expression to model the wining conditions. Here it is necessary to decide upon the state of the fields of the board, i.e. given a position it is asked, if a white or black stone occupies the field or if it is blank. As opposed to Tic-Tac-Toe the moves do not explicitly code the field that is involved, but only the column. Therefore all moves have to be evaluated with respect to a certain column, to decide which fields of the column are occupied. The order of stones below the position in question does not matter and for this all permutations have to be considered.

4. Experimental Results

Experiments for each game were carried out. The n Queens problem was scaled up to $n = 10$, for Tic-Tac-Toe the two different coding schemes are compared and finally the model of Connect Four was successively built until the expression became too large. Details are given below.

All experiments were carried out on an Athlon 1,8+GHz with 512 MByte under Linux. CUDD was used as the underlying BDD package [15]. This package amongst other features provides the possibility to activate automatic sifting [13]. If the number of nodes exceeds a certain threshold, sifting is executed. In our experiments we used automatic sifting with the default settings of the package.

The tables list the number of nodes, the peak number of nodes and the CPU-time. The number of nodes (column '#nodes') is the number of nodes needed in the decision diagram to represent the final expression win. The peak number of nodes (column 'peak #') is the number of nodes at the maximum, including the nodes needed to represent the expression all. When carrying out an operation on

Table 1. n **Queens problem**

n	solu.	without sifting			with sifting		
		#nodes	peak #	CPU	#nodes	peak #	CPU
3	0	1	113	0 s	1	113	0 s
4	2	30	475	0 s	30	475	0 s
5	10	167	2392	0 s	167	2392	0 s
6	4	130	12121	0.02 s	129	6412	0.13 s
7	40	1099	63779	0.1 s	1223	23516	1.26 s
8	92	2451	333000	0.71 s	2984	92693	12.28 s
9	352	9557	1737958	18.86 s	13111	323904	255.09 s
10	724	25945	8918190	562.99 s	32582	1586744	3600.8 s
11				>15h			>15h

Table 2. Tic Tac Toe

Version				
#var.	sifting	#nodes	peak #	CPU
81	no	47219	152438	1.12 s
81	yes	108717	263511	102.46 s
36	no	22417	75054	0.49 s
36	yes	29080	76437	7.04 s

two functions, building a sub-expression or during sifting, the number of nodes can temporarily surpass the number of nodes that are finally needed.

4.1 n Queens

The n Queens problem has been used as a benchmark in [11] and [12] already. Its suitability for this purpose is emphasized by the first experiment. In Table 1 the results are given. As can be seen calculating the solution of the problems with n less than seven is very fast. The threshold for automatic sifting is reached for n greater than five only. In this case a lot more time is needed to carry out the minimization of the binary decision diagram but still the problem is solvable. After exceeding the value ten for n swapping started, so the runtime dramatically increased. Indeed no result was generated within 15 hours. This scalability makes the problem perfectly suitable to compare the influence of the size of a problem on different techniques.

4.2 Tic-Tac-Toe

On Tic-Tac-Toe the influence of different coding styles on the resulting model is compared and it turns out that the natural coding of moves in rows and columns is more efficient than the one-hot scheme. Furthermore it can be observed that the application of sifting on the model of the sequential problem is of no use. Results of the two different coding schemes for Tic-Tac-Toe are shown in Table 2.

In the one-hot coding scheme the state - circle, ’x’ or blank - is determined by nine variables, since one variable per move corresponds to that field. Opposing to that in the binary coding scheme all four variables for one move and all 36 variables for the game have to be evaluated to decide on the state of a field. Therefore one could argue that the one-hot coding scheme results in a less complex function. But the results of the experiment outrule this argument. The coding of a move by four input variables is more efficient: Only 36 instead of 81 variables are needed to code the game and the resulting binary decision diagram is smaller.

In both cases the problem is tractable without applying sifting. Therefore the activation of sifting results in a higher runtime, even worse, sifting as being a heuristic minimization approach leads to a larger resulting decision diagram. So the natural variable order derived from the sequential assignment of values to inputs is an efficient one. This can intuitively be conceived. The evaluation of the early moves can lead to a victory which corresponds to a constant node in the decision diagram. Opposing to that the evaluation of the later moves can never lead to a constant node before the early variables are evaluated. Overall more nodes in the decision diagram are needed, if the later variables are evaluated first.

4.3 Connect Four

Connect Four is a more complex problem than Tic-Tac-Toe, the function to model the game could not be fully built. As can be seen in Table 3 the number of nodes in

Table 3. Connect Four

sift	move	#nodes	peak #	CPU
no	6	1	1022	0s
	7	500	14308	0.03s
	8	2714	24528	0.12s
	9	11200	79716	0.38s
	10	34822	251412	1.28s
	11	147843	1011780	3.9s
	12	463083	2544780	12.83s
	13	1673493	6013448	54.47s
	14	4811513	17586576	235.53s
	15			>15h
yes	6	1	1022	0s
	7	500	14308	0.02s
	8	2594	24528	0.23s
	9	11054	59276	2.12s
	10	34562	175784	23.63s
	11	147438	683718	195.63s
	12	463728	2226938	596.11s
	13	1687123	5967458	5294.27s
	14			>15h

the decision diagram drastically increases with the number of moves that are modeled. While up to the 14th move the physical memory was sufficient to model the game, swapping started when modeling the 15th move. When automatic sifting was applied the 14th move could not be modeled within 15 hours. Again sifting is of no use and the sequential variable order is an efficient one. Here the same argument as for Tic-Tac-Toe applies.

5. Conclusions and Discussion

We proposed games as benchmarks for comparing techniques in the domain of multi-valued logic. In particular three games were investigated. The scalability of the n Queens problem shows its suitability for benchmarking (as said above it has already been used by other authors for benchmarking). The importance of the coding scheme for a certain problem was shown by comparing two schemes for Tic-Tac-Toe. Finally, in the case of Connect Four the sequential depth was shown to be the limit of the model.

Nonetheless it is possible to symbolically represent the full solution space of non-trivial multi-valued problems. This leads to the availability of a fast evaluation of candidate solutions. By incorporating this into a learning algorithm the quality of a particular solution can be measured efficiently. It is still to investigate, if a good heuristic for a small instance of a problem leads to a good heuristic for larger instances, also. Focus of future work is to evaluate different problems with respect to these questions.

References

[1] L. V. Allis. A knowledge-based approach to connect-four. the game is solved: White wins. Master's thesis, Faculty of Mathmatics and Computer Science, Vrije Universiteit, Amsterdam, Oct. 1988.

[2] P. Ashar and S. Malik. Fast functional simulation using branching programs. In *Int'l Conf. on CAD*, pages 408–412, 1995.

[3] F. Brglez and R. Drechsler. Design of experiments in CAD: Context and new data sets for ISCAS'99. In *Int'l Symp. Circ. and Systems*, volume VI, pages 424–427, 1999.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] O. Coudert. Solving graph optimization problems with ZBDDs. In *Design, Automation and Test in Europe*, pages 224–228, 1997.

[6] N. Drechsler, F. Schmiedle, D. Große, and R. Drechsler. Heuristic learning based on genetic programming. In *EuroGP*, volume 2038 of *LNCS*, pages 1–10. Springer Verlag, 2001.

[7] M. Löbbing, , and I. Wegener. The number of knight's tours equals 33, 439, 123, 484, 294 - counting with binary decision diagrams. *The Electronic Journal of Combinatorics*, 3(1):R5, 1996.

[8] M. Löbbing, O. Schröer, and I. Wegener. The theory of zero-suppresed BDDs and the number of knight's tours. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 38–45, 1995.

[9] D. M. Miller and R. Drechsler. On the construction of multiple-valued decision diagrams. In *Int'l Symp. on Multi-Valued Logic*, pages 245–253, 2002.

[10] S. Minato. Zero-suppressed BDDs for set manipulation in combinational problems. In *Design Automation Conf.*, pages 272–277, 1993.

[11] S. Minato. Calculation of unate cube set algebra using zero-suppressed BDDs. In *Design Automation Conf.*, pages 420–424, 1994.

[12] S. Minato and S. Ishihara. Streaming BDD manipulation for large-scale combinatorial problems. In *Design, Automation and Test in Europe*, pages 224–228, 2001.

[13] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[14] T. Sasao. Compact SOP representations for multiple-output functions - an encoding method using multiple-valued logic. In *Int'l Symp. on Multi-Valued Logic*, pages 207–212, 2001.

[15] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.