On the Complexity of BDDs for State Space Search: A Case Study in Connect Four

Stefan Edelkamp and **Peter Kissmann**
TZI Universität Bremen, Germany
{edelkamp, kissmann}@tzi.de

Abstract

Symbolic search using BDDs usually saves huge amounts of memory, while in some domains its savings are moderate at best. It is an open problem to determine if BDDs work well for a certain domain.

Motivated by finding evidences for BDD growths for state space search, in this paper we are concerned with symbolic search in the domain of CONNECT FOUR. We prove that there is a variable ordering for which the set of all possible states – when continuing after a terminal state has been reached – can be represented by polynomial sized BDDs, whereas the termination criterion leads to an exponential number of nodes in the BDD given any variable ordering.

Introduction

Binary Decision Diagrams (BDDs) (Bryant 1986) can be a very efficient data structure for storing large sets of states, and we can use them to perform set-based search. For some domains they can save an exponential amount of memory (compared to explicit representation), in others the saving is only a small linear factor, while in most domains its savings are dependent on the variable ordering.

Thus, when trying to work on a domain with BDDs, a question that immediately arises is whether the BDDs will be of any help. For some well-known planning benchmarks such an analysis has already been done. For example, Hung (1997) has proven that BDDs for representing the exact set of reachable states in permutation games such as most SLIDING TILES PUZZLES, especially the well-known $(n^2 - 1)$-PUZZLE, or the BLOCKSWORLD domain, are of size exponential in the number of variables of the permutation, no matter what variable ordering is chosen. On the other hand, we have shown that, given a good variable ordering, BDDs can save an exponential amount of memory in the domain of GRIPPER (Edelkamp and Kissmann 2008a).

The game of CONNECT FOUR belongs to the broader class of k-in-a-row games such as TIC-TAC-TOE or GOMOKU. On a $w \times h$ board (w being the width and h the height) the two players (red and yellow) take turns placing their pieces on the board. In CONNECT FOUR gravity is important, so that the pieces fall as far to the bottom as possible. Thus, in each state at most w moves are possible (placing the piece in one of the columns, if it is not yet completely filled). The goal is to construct a line of (at least) four pieces of the own color (cf. Figure 1).

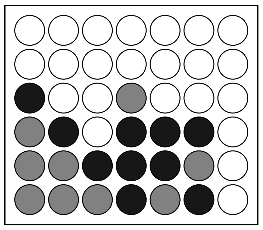


Figure 1: The CONNECT FOUR domain.

The default instance of CONNECT FOUR using a 7×6 board has been solved independently only a few days apart by Allis (1988) and Allen[1] (2011), but at that time the precise number of reachable states was unknown – Allis provided only a rough estimate of about $7 \cdot 10^{13}$ states. In an earlier experiment we have determined the exact number with symbolic search using BDDs in about 15 hours (Edelkamp and Kissmann 2008b) – there are exactly $4{,}531{,}985{,}219{,}092$ states reachable. Later, Tromp came up with the same number using explicit search, which took about $10{,}000$ hours.[2] Given these runtimes it seems likely that BDDs are good for generating the set of reachable states.

In this paper we will show that BDDs really are good for representing the reachable states if we continue search after finding a terminal state. When stopping at terminal states, things are more difficult. This is because, in contrast to all the previously mentioned problems, the termination criterion in CONNECT FOUR is complex and needs a BDD with an exponential number of nodes to be represented.

In the following we will give an introduction to BDDs and the importance of the variable ordering as well as to symbolic search. Then, we will prove that the reachable states of each BFS layer can be represented by a BDD polynomial in the board size, while representing even a reduced termination criterion already results in a BDD of size exponential in the smaller of the board's edges.

[1]Reported in rec.games.programmer on October 1st 1988.

[2]`http://homepages.cwi.nl/~tromp/c4/c4.html`

Binary Decision Diagrams

Binary Decision Diagrams (BDDs) (Bryant 1986) are a memory-efficient data structure used to represent Boolean functions as well as to perform set-based search. In short, a BDD is a directed acyclic graph with one root and two terminal nodes, the 0- and the 1-sink. Each internal node corresponds to a binary variable and has two successors, one (along the low edge) representing that the current variable is false ($\perp$) and the other (along the high edge) representing that it is true ($\top$). For any assignment of the variables derived from a path from the root to the 1-sink the represented function will be evaluated to $\top$.

Bryant proposed to use a fixed variable ordering, for which he also provided two reduction rules (eliminating nodes with the same low and high successors and merging two nodes representing the same variable that share the same low successor as well as the same high successor). These BDDs were called reduced ordered binary decision diagrams (ROBDDs). Whenever we mention BDDs in this paper we actually refer to ROBDDs.

The Variable Ordering

For many Boolean formulas the corresponding BDD can be of polynomial or even linear size, given a good variable ordering π, although the number of satisfying assignments (i. e., the number of states) might be exponential in n. One such case is the Disjoint Quadratic Form (DQF).

Definition 1 (Disjoint Quadratic Form). *The* disjoint quadratic form *DQF_n is defined as*

$$DQF_n\left(x_1,\ldots,x_n,y_1,\ldots,y_n\right) := x_1y_1 \vee x_2y_2 \vee \ldots \vee x_ny_n.$$

Proposition 1 (Linear Size of a BDD for DQF). *Given the variable ordering $\pi = (x_1, y_1, \ldots, x_n, y_n)$ the BDD representing the DQF_n contains $2n+2$ nodes.*

On the other hand, a bad variable ordering might result in a BDD of size exponential in n. For the DQF we can find such an ordering as well.

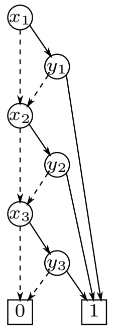
Proposition 2 (Exponential Size of a BDD for DQF). *Given the variable ordering $\pi = (x_1, \ldots, x_n, y_1, \ldots, y_n)$ the BDD representing the DQF_n contains 2^{n+1} nodes.*

Figure 2 illustrates both cases for the DQF$_3$.

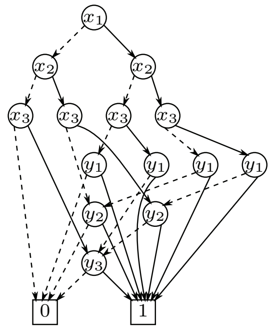
Furthermore, there are some functions for which any BDD contains $\Omega\left(2^n\right)$ nodes, e. g., integer multiplication, for which Bryant (1986) proved that for any variable ordering there is an output bit for which the BDD contains at least $2^{n/8}$ BDD nodes.

Finally, for some functions any variable ordering results in a polynomial sized BDD. An example for this would be any symmetric function, for which a BDD contains at most $\mathcal{O}\left(n^2\right)$ nodes.

Nevertheless, for many functions there are good variable orderings and finding these might be crucial. Yet, finding an ordering that minimizes the number of BDD nodes is co-NP-complete (Bryant 1986). Furthermore, the decision whether a variable ordering can be found so that the resulting BDD contains at most s nodes, with s being specified beforehand, is NP-complete (Bollig and Wegener 1996).



(a) Good variable ordering.



(b) Bad variable ordering.

Figure 2: Two possible BDDs representing DQF$_3$. Solid lines represent high edges, dashed ones low edges.

Additionally, Sieling (2002) proved that there is no polynomial time algorithm that can calculate a variable ordering so that the resulting BDD contains at most c times as many nodes as the one with optimal variable ordering for any constant $c > 1$, so that we cannot expect to find good variable orderings using reasonable resources. This shows that using heuristics to calculate a variable ordering (Fujita, Fujisawa, and Kawato 1988; Malik et al. 1988; Butler et al. 1991) is a plausible approach.

Symbolic Search

State space search is an important topic in many areas of artificial intelligence. No matter if we want to find a shortest path from one location to another, some plan transforming a given initial state to another state satisfying a goal condition, or finding good moves for a game – in all these cases search is of high importance.

In explicit search we need to come up with good data structures to represent the states efficiently or algorithms for fast transformation of states. In case of symbolic search (McMillan 1993) with BDDs the data structure to use is already given. As to efficient algorithms, here we need to handle things in a different manner compared to explicit search. While explicit search is concerned with expansion of single states and calculation of successors of a single state, in symbolic search we handle sets of states. The assignments satisfying a Boolean formula can be seen as sets of states, as well. Similarly, we can represent any state we encounter in a state space search as a Boolean formula with one satisfying assignment. To achieve this, we represent any state as a conjunction of (binary) variables. Thus, a set of states can be seen as the disjunction of such a conjunction of variables, so that we can easily represent it by a BDD.

To perform symbolic search we need two sets of binary variables. One set, S, stores the current states, while the other set, S', stores the successor states. With these, we can represent a transition relation, which connects the current states with their successors.

For a given set of actions $\mathcal{A}$, any action $a \in \mathcal{A}$ typically consists of a precondition and a number of effects to specify the changes to a state satisfying the precondition and

Algorithm 1 Symbolic Breadth-First Search (SBFS)

```
Input: Search Problem $\mathcal{P} = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{T} \rangle$.
 1: $reach_0 \leftarrow \mathcal{I}$;
 2: $layer \leftarrow 0$;
 3: while $reach_{layer} \neq \bot$ do
 4:     $current \leftarrow reach_{layer} \land \neg\mathcal{T}$;
 5:     $reach_{layer+1} \leftarrow image\,(current)$;
 6:     for $0 \leq i \leq layer$ do
 7:         $reach_{layer+1} \leftarrow reach_{layer+1} \land \neg reach_i$;
 8:     end for
 9:     $layer \leftarrow layer + 1$;
10: end while
```

thus the successor state. For each $a = (pre_a, eff_a) \in \mathcal{A}$ we can generate one transition relation $trans_a\,(S, S') := pre_a\,(S) \land eff_a\,(S') \land frame_a\,(S, S')$ with $frame_a$ being the frame for action a, which determines what parts of the current state do not change when applying a. These transition relations can be combined to generate a single (monolithic) transition relation *trans* by calculating the conjunction of all the smaller transition relations:

$$trans\,(S, S') := \bigvee_{a \in \mathcal{A}} trans_a\,(S, S')$$

To calculate the successors of a set of states *current*, the *image* operator is defined as

$$(\exists S. trans\,(S, S') \land current\,(S))\,[S' \rightarrow S]\,.$$

Given a set of states *current*, specified in S, it calculates all the successor states, specified in S'. Afterwards, all variables of S are replaced by those of S' (denoted by $[S' \rightarrow S]$), so that the successor states finally are represented in S and we can continue the search.

The combination of conjunction and existential quantification used in the *image* operator, the so-called relational product, is one of the bottlenecks in symbolic search, as its calculation is NP-complete (McMillan 1993). Most BDD packages contain implementations of the relational product which are much more efficient than the manual execution of conjunction followed by the existential quantification.

Given an implementation of the *image* operator, performing symbolic breadth-first search (BFS) is straight-forward (cf. Algorithm 1). For any search problem $\mathcal{P} = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{T} \rangle$ with $\mathcal{V}$ being a set of (Boolean) variables, $\mathcal{A}$ a set of actions, $\mathcal{I}$ the initial state and $\mathcal{T}$ the set of terminal states, all we need to do is to apply the *image* operator first to $\mathcal{I}$ and afterwards to the last generated successor state set. The search ends when a fix-point is reached (line 3). For this we must remove any terminal states before expanding a set of states (line 4) and all duplicate states that might reside in a newly generated BFS layer (lines 6 – 8).

BDD Sizes for CONNECT FOUR

In this section we will analyze the behavior of BDDs in the context of CONNECT FOUR. First, we will prove that the calculation of the set of reachable states can be done using polynomial sized BDDs if we continue the search after

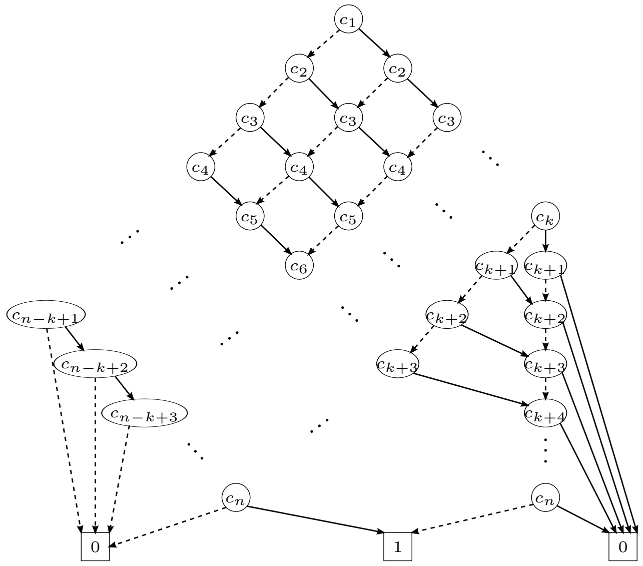


Figure 3: Critical part of the BDD for representing k cells being occupied by pieces of one color. Solid edges correspond to the cell being occupied by a piece of the color in question, dashed ones to the other case.

reaching terminal states. Afterwards, we will have a closer look at the termination criterion and prove that even for a simpler version of it we already need exponentially many BDD nodes.

Polynomial Upper Bound for Representing the Reachable States

CONNECT FOUR is a game where in each state l pieces are placed on the $n = wh$ positions of the game board, so that its search space is exponential. Nevertheless, we can find a variable ordering, so that the reachable states can be represented by a number of polynomial sized BDDs (one for each BFS layer) if we continue after a terminal state has been reached, i. e., if we omit the removal o terminal states (line 4 of Algorithm 1).

Theorem 3 (Polynomial Bound for BFS Layers without Termination in CONNECT FOUR). *There is a binary state encoding and a variable ordering for which the BDD size for the characteristic function of any BFS layer in* CONNECT FOUR *is polynomial in $n = wh$, with w being the width and h the height of the board, if we do not cut off the search after reaching a terminal state.*

Proof. We use $2n + 1$ bits to encode a state in CONNECT FOUR, two for each cell (red / yellow / none) and one for the active player. If we organize the encoding in a columnwise manner we need to calculate the conjunction of three polynomial sized BDDs for each layer l.

Two BDDs are similar to the one in Figure 3. The first represents the fact that we have $k = \lceil l/2 \rceil$ red pieces on the board, the second one that we have $k = \lfloor l/2 \rfloor$ yellow pieces on the board. As in this game the cells can also be empty we need to replace each node by two, the first one distinguishing

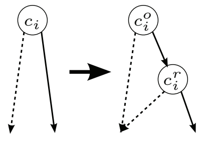


Figure 4: Replacement of a node representing that cell i is occupied by a red piece by two nodes denoting that the cell is occupied (c_i^o) and the occupying piece is red (c_i^r).

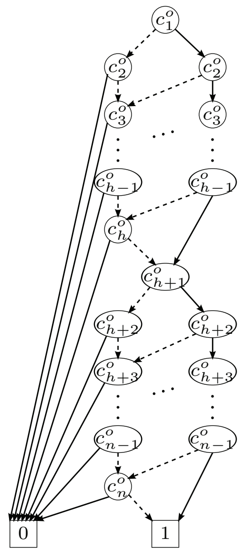


Figure 5: BDD for representing the fact that all pieces are located at the bottom. Each node c_i^o represents the decision for cell i if it is occupied.

between the cases that a cell is empty or not and the second one between the colors. If the cell is empty we insert an edge to a node for the next cell; if it is not we distinguish between the two colors (cf. Figure 4 for red pieces; for yellow pieces, the low and high edges of the lower node are swapped). Due to the replacement of each cell by two BDD nodes, each BDD has a size of $\mathcal{O}(2kn) = \mathcal{O}(ln)$.

The third BDD represents the gravity, i. e., the fact that all pieces are located at the bottom of the columns. For this a column-wise variable ordering is important. Starting at the lowest cell of a column, if one is empty all the following cells of this column are necessarily empty as well. If one is not empty the others can still be empty or not. In this BDD it is only relevant if a cell is occupied, so that the actual color of a piece occupying a cell does not matter, which keeps the BDD smaller. Thus, we need at most two BDD nodes for each cell, resulting in a total of $\mathcal{O}(2n) = \mathcal{O}(n)$ BDD nodes (cf. Figure 5).

Calculating the conjunction of two BDDs of sizes s_1 and s_2 results in a new BDD of size $\mathcal{O}(s_1 s_2)$, so that the conjunction of three polynomial BDDs is still polynomial. In this case, the total BDD for each layer l is of size $\mathcal{O}\left(l^2 n^3\right)$. As l is at most n, we arrive at a total size of $\mathcal{O}\left(n^5\right)$. □

Exponential Lower Bound for the Termination Criterion

In the domains analyzed before, the termination criterion was always trivial to model as a BDD. E. g., in the $\left(n^2-1\right)$-PUZZLE and BLOCKSWORLD only one pattern is correct, while in GRIPPER all balls must end up in room B.

In contrast to these, integrating the terminal states in the set of reachable states (i. e., not continuing search from terminal states) in CONNECT FOUR is a lot more difficult. Here, the termination criterion is fulfilled if one player has succeeded in placing four own pieces in a row (horizontally, diagonally, or vertically), which results in a rather complex Boolean formula.

In the following we will prove that even for a simplified version of this termination criterion, i. e., one that is fulfilled when four adjacent cells in a horizontal or vertical line are occupied, any BDD is of exponential size for a CONNECT FOUR problem encoded in the way proposed in Theorem 3.

To prove this, we can reduce the termination criterion even further.

Definition 2 (Reduced Termination Criterion). *The reduced termination criterion $\mathcal{T}_r$ is the reduction of the full termination criterion to the case where only two horizontally or vertically adjacent cells must be occupied, i. e., $\mathcal{T}_r = \bigvee_{1\le i\le w, 1\le j\le h-1}\left(x_{i,j}\wedge x_{i,j+1}\right) \vee \bigvee_{1\le i\le w-1, 1\le j\le h}\left(x_{i,j}\wedge x_{i+1,j}\right)$ with $x_{i,j}$ specifying that the cell at column i and row j is occupied.*

We can represent the game board by a grid graph $G = (V, E)$, with V representing the wh cells of the board, with edges only between horizontally or vertically adjacent nodes. The reduced termination criterion corresponds to the fact that two nodes connected by an edge are occupied.

Additionally, we make use of the disjoint quadratic form DQF, which we introduced in Definition 1.

Lemma 4 (Exponential Bound for $\mathcal{T}_r$). *For* CONNECT FOUR *in the cell-wise encoding from Theorem 3 with a board size of $w \times h$ the size of the BDD for representing $\mathcal{T}_r$ is exponential in* $\min(w,h)$*, independent of the chosen variable ordering.*

Proof. To prove this we show that any variable ordering can be split in two, so that $\Omega(\min(w,h))$ variables are in the one part and connected to $\Omega(\min(w,h))$ variables in the other part. Retaining at most one connection for each variable, we will show that $\Omega(\min(w,h)/7)$ edges still connect the two sets. These represent a DQF with a bad variable ordering, which will give us the exponential bound.

Given any ordering π on the variables we divide it into two sets π_1 and π_2 with π_1 being the first $\lceil wh/2\rceil$ variables of π and π_2 the remaining $\lfloor wh/2\rfloor$ variables. These sets correspond to a partitioning of the grid graph G into two sets.

The minimal number of crossing edges between these partitions is in $\Omega(\min(w,h))$. It is achieved when each partition is connected and forms a rectangle of size $(\max(w,h)/2)\cdot\min(w,h)$, so that the cut between the two partitions crosses the grid in the middle but orthogonal to the longer edge and thus crosses $\min(w,h)$ edges.

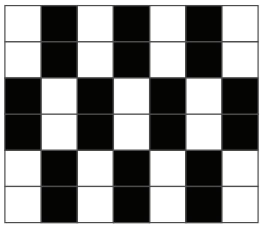


Figure 6: Coloring of the game board in Lemma 5.

We might view the two partitions along with the connecting edges as a bipartite graph $G' = (V_1, V_2, E')$, $|E'| = \min(w, h)$, for which we want to find a (not necessarily maximal) matching. To find one, we choose one node from V_1 and a connecting edge along with the corresponding node from V_2. The other edges starting at one of these nodes are removed (there are at most six of them, as each node can be connected to at most four neighbors, one of which is chosen). This we repeat until we cannot remove any more edges. As we choose one edge in each step and remove at most six, the final matching M contains at least $|E'|/7 = \min(w, h)/7$ edges.

All variables that are not part of the matching are set to $\perp$[3], while the others are retained. Thus, the corresponding formula for $\mathcal{T}_r$ is the DQF over the two sets of variables representing the nodes that are part of the matching. All variables for the one set ($V_1 \cap M$) appear before all variables of the other set ($V_2 \cap M$), so that we have a bad ordering for these variables and the corresponding BDD is of exponential size. Thus, according to Proposition 2 the BDD will contain at least $\Omega\left(2^{\min(w,h)/7}\right) = \Omega\left(\left(2^{1/7}\right)^{\min(w,h)}\right) = \Omega\left(1.1^{\min(w,h)}\right)$ nodes. □

This result immediately implies that for square boards the number of BDD nodes is exponential in the square root of the number of cells, i. e., the BDD for representing $\mathcal{T}_r$ contains at least $\Omega\left(1.1^{\sqrt{n}}\right)$ nodes.

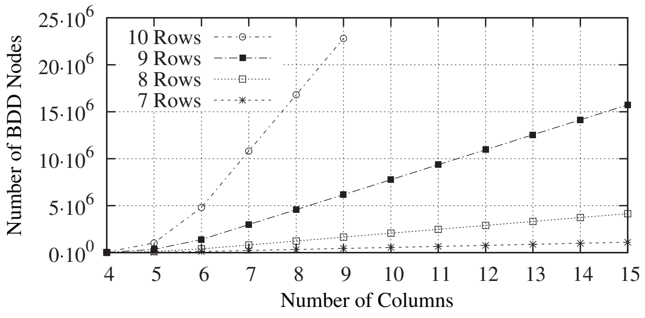
With this result we can show that the case of four adjacent cells in a horizontal or vertical line being occupied results in a BDD of exponential size as well.

Lemma 5 (Exponential Bound for Four Pieces in a Horizontal or Vertical Line). *Any BDD representing the part of $\mathcal{T}$ that states that four cells in a horizontal or vertical line must be occupied is exponential in* $\min(w, h)$ *for a board of size* $w \times h$.

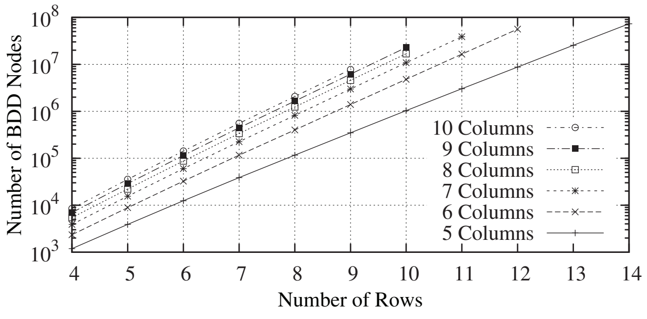
Proof. We can devise a two-coloring of the game board as shown in Figure 6. Using this, any line of four pieces resides on two white and two black cells. If we replace all variables representing the black cells by $\top$ we can reduce this part of $\mathcal{T}$ to $\mathcal{T}_r$. As this replacement does not increase the BDD size, the BDD is exponential in $\min(w, h)$ as well.

□

[3]Replacing a variable by a constant does not increase the number of BDD nodes.



(a) Fixed heights.



(b) Fixed widths (log-scale).

Figure 7: Number of BDD nodes for the termination criterion for one player. The variable ordering is according to the columns.

In a similar fashion we can show that any BDD for the fact that two (or four) diagonally adjacent cells are occupied is exponential for any variable ordering. The combination of all these results illustrates that any BDD representing the termination criterion in CONNECT FOUR is of exponential size, no matter what variable ordering is chosen.

Experimental Evaluation

For a column-wise variable ordering we have evaluated the number of BDD nodes for representing the termination criterion of four pieces of one player being placed in a horizontal or vertical line for several board sizes (cf. Figure 7). For a fixed number of rows (cf. Figure 7a) the growth of the number of BDD nodes is linear in the number of columns, while for a fixed number of columns (cf. Figure 7b) it is exponential in the number of rows. Though it first might seem like a contradiction to our result, it actually shows that it holds for the chosen variable ordering.

In this variable ordering the partitions induced by the split of the variable ordering cut h edges. Thus, the resulting BDD's size is exponential in h. For the case of a fixed height, h does not change, so that the size increases by a constant amount of nodes. On the other hand, when fixing the width, the size changes exponentially in the height, no matter if it is smaller or larger than the width. This happens because for the cases that the height is actually larger than the width this variable ordering is not the best choice – in those cases it should have been in a row-wise manner.

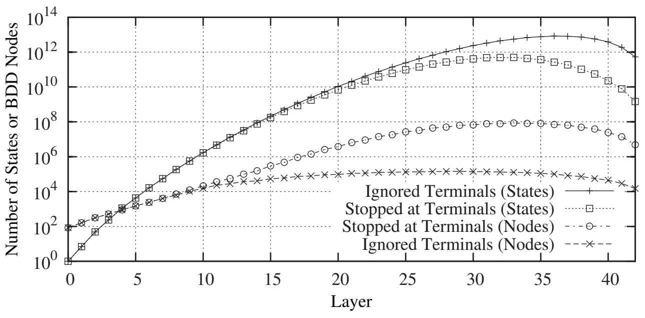


Figure 8: Comparison of the number of states and BDD nodes for different layers with a 7×6 board when ignoring terminal states and when stopping at terminal states. The variable ordering is according to the columns.

Table 1: Classification of some benchmarks with respect to the BDD sizes.

	Polynomial Reachability	Exponential Reachability
Polynomial Termination Criterion	GRIPPER	BLOCKSWORLD SLIDING TILES PUZZLE $(n^2 - 1)$-PUZZLE
Exponential Termination Criterion	CONNECT FOUR TIC-TAC-TOE GOMOKU	

Furthermore, we have evaluated the default instance of 7×6 along with the full termination criterion in two complete searches, one stopping at terminal states and one continuing after them (cf. Figure 8). Again we used a column-wise variable ordering. We can see that the number of states in the first is smaller than in the second case, while the number of BDD nodes needed to represent these states is larger in the first case. Thus, for this variable ordering we can clearly see that the use of the termination criterion increases the sizes of the BDDs.

Comparing the runtimes we arrive at a similar conclusion: On our machine (Intel Core $i7$ 920 with 2.67 GHz and 24 GB RAM) the runtime decreased from $4{:}48$ hours to 11 minutes when ignoring the termination criterion.

Conclusion

In this paper we have determined bounds for the BDD sizes for representing the reachable states (when ignoring terminal states) as well as for representing the termination criterion in CONNECT FOUR. Though we have only analyzed one state encoding (a cell-wise encoding), we believe that the exponential lower bound also holds for any other encoding. Another plausible encoding we see is a piece-wise encoding, but for this the encoding of one state increases from $2n + 1$ bits to $n\left(\lceil \log n \rceil + 1\right)$ bits. Furthermore, in our previous analysis of SOKOBAN (Edelkamp and Kissmann 2008a) we saw that at least for that domain the reachability analysis becomes much more complex.

With the new results we can now classify several games with respect to the complexity of BDDs (cf. Table 1).

In the domains analyzed so far, the BDDs were either exponential for any variable ordering or polynomial for at least one variable ordering. In CONNECT FOUR we have identified a domain, for which BDDs can be polynomial, but only if we do not construct the BDD for the termination criterion, which itself is exponential.

The lower bound results of this paper are surprisingly general. We have proven that a BDD for representing the the case of two adjacent cells being occupied – no matter by what color – is of exponential size. This might come in handy in the analysis of other domains. For example, in CHESS part of a checkmate is that all cells adjacent to a king are either occupied by its own figures or guarded by the opponent. Furthermore, the capture rule for pawns might be formulated in a similar way, so that it is a possible candidate for filling the final bucket in Table 1.

Acknowledgments

Thanks to Deutsche Forschungsgesellschaft (DFG) for support in the project ED $74/11$-1.

References

Allen, J. D. 2011. *The Complete Book of Connect 4: History, Strategy, Puzzles*. Puzzlewright.

Allis, L. V. 1988. A knowledge-based approach of connect-four. Master's thesis, Vrije Universiteit Amsterdam.

Bollig, B., and Wegener, I. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers* 45(9):993–1002.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Butler, K. M.; Ross, D. E.; Kapur, R.; and Mercer, M. R. 1991. Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. In *DAC*, 417–420. ACM Press.

Edelkamp, S., and Kissmann, P. 2008a. Limits and possibilities of BDDs in state space search. In Fox, D., and Gomes, C. P., eds., *AAAI*, 1452–1453. AAAI Press.

Edelkamp, S., and Kissmann, P. 2008b. Symbolic classification of general two-player games. In Dengel, A.; Berns, K.; Breuel, T. M.; Bomarius, F.; and Roth-Berghofer, T., eds., *KI*, volume 5243 of *LNCS*, 185–192. Springer.

Fujita, M.; Fujisawa, H.; and Kawato, N. 1988. Evaluation and improvements of Boolean comparison method based on binary decision diagrams. In *ICCAD*, 2–5. IEEE.

Hung, W. 1997. Exploiting symmetry for formal verification. Master's thesis, University of Texas at Austin.

Malik, S.; Wang, A. R.; Brayton, R. K.; and Sangiovanni-Vincentelli, A. 1988. Logic verification using binary decision diagrams in a logic synthesis environment. In *ICCAD*, 6–9. IEEE.

McMillan, K. L. 1993. *Symbolic Model Checking*. Kluwer Academic Publishers.

Sieling, D. 2002. The nonapproximability of OBDD minimization. *Information and Computation* 172(2):103–138.