Symbolic Exploration in Two-Player Games: Preliminary Results

Stefan Edelkamp

Institut für Informatik
Albert-Ludwigs-Universität
Georges-Köhler-Allee, Geb. 51
D-79110 Freiburg
eMail: edelkamp@informatik.uni-freiburg.de

Abstract

In this paper symbolic exploration with binary decision diagrams (BDDs) is applied to two-player games to improve main memory consumption for reachability analysis and game-theoretical classification, since BDDs provide a compact representation for large sets of game positions. A number of examples are evaluated: Tic-Tac-Toe, Nim, Hex, and Four Connect. In Chess we restrict the considerations to the creation of endgame databases. The results are preliminary, but the study puts forth the idea that BDDs are widely applicable in game playing and provides a universal tool for people interested in quickly solving practical problems.

Introduction

BDDs have encountered AI for different purposes. The most apparent area is (especially non-deterministic) planning (Cimatti, Roveri, & Traverso 1998), since the spaces of planning problems tend to be very large. AI-search can be casted as Model Checking (Clarke, Grumberg, & Peled 2000), where it has been observed that BDDs can be used to concisely express sets of states in a state transition system. Symbolic reachability analysis traverses each state within the system and applies to transition systems with more than \(10^{20}\) states. Even verifying temporal properties as specified in different logics according to additionally labeled states is possible for moderate system sizes.

BDDs have also been applied to single-agent problems like the \((n^2 - 1)\)-Puzzle and Sokoban (Edelkamp & Reffel 1998). The applied search algorithm BDDA* enjoys recent research interests of the groups Hansen, Zhou, and Feng (ADDA*), or Jensen, Bryant and Veloso (SetA*). However, due to the lack of refined symbolic information the results for \((n^2 - 1)\)-Puzzle and Sokoban are currently too weak to compete with heuristic single state-space search techniques.

As game playing in AI contributes to both AI-planning and AI-search, this paper considers two- and multi-player games in exploring large sets of game positions with moderate node sizes of the corresponding BDD data structure. It is structured as follows. In the next two sections we briefly address BDD basics and two-player zero-sum games with complete information. Afterwards we introduce the symbolic exploration technique with respect to the example of Tic-Tac-Toe. We show how to encode the problem and how to perform reachability analysis and game-theoretical classification. In the experimental section we additionally address games like Nim, Hex, and Four Connect, and explain how to compute symbolic endgame databases for Chess.

BDDs

Binary decision diagrams (Bryant 1985) have been introduced in the context of hardware verification and are a graphical representation of Boolean functions. More precisely, a binary decision diagram (BDD) to represent a Boolean function \(f\) is a directed rooted acyclic graph with one or two terminal nodes, labeled with 0 and 1, and internal nodes with out-degree two. A Boolean variable from the variable set of the represented function \(f\) is associated to every internal node. The outgoing edges of node \(k\) are labeled with \(low(k)\) and \(high(k)\). The interpretation of a BDD for \(f\) is \(f_{low}\) if the variable at the root node is zero and \(f_{high}\) otherwise, where \(f_{low}\) and \(f_{high}\) are itself interpreted as BDDs.

For a compact representation of \(f\) two rules for reducing the graph representation are available. Rule 1 deletes a node with all outgoing edges if there exists another node with the same labeling and same successors. Rule 2 deletes a node \(k\) if \(low(k)\) is equal to \(high(k)\). A BDD is reduced if neither Rule 1 nor Rule 2 can be applied anymore. It is ordered, if on every path in the graph a total ordering of the variables is preserved. Since reduced, ordered BDDs are a unique representation of Boolean functions, with the term BDD we always refer to reduced and ordered BDDs.

The main advantage with respect to a truth table is that in practice BDDs tend to have tractable (polynomial) size even if the represented set of all satisfying assignments is intractable (exponential). This is due to the fact that a directed acyclic graph commonly represents exponentially many paths. A further advantage is that given two BDD-representations for functions \(f\)
and $g$ the logical operations $f \land g$, $f \lor g$ and $\neg f$ can be executed efficiently and for the existential ($\exists$) and universal ($\forall$) quantification the efficient graph representation positively influences the execution time.

### Two-Player Zero-Sum Games

A two-player zero-sum game (with perfect information) is given by a set of states $S$, move-rules to modify states and two players, called Player 0 and Player 1. Since one player is active at a time, the entire state space of the game is $Q = S \times \{0, 1\}$. A game has an initial state and some predicate goal to determine whether the game has come to an end. We assume that every path from the initial state to a final one is finite. For the set of goal states $G = \{s \in Q : \text{goal}(s)\}$ we define an evaluation function $v : G \to \{-1, 0, 1\}$, $-1$ for a losing position, $1$ for a winning position, and $0$ for a drawing one. This function is extended to $\hat{v} : Q \to \{-1, 0, 1\}$ asserting a game theoretical value to each state in the game.

Let $L(i)$ be the set of lost positions for Player $i$, $i \in \{0, 1\}$. The set $L(1)$ can be recursively calculated as follows: i) All lost position $s$ for Player 1 are contained in $L(1)$, i.e., $\{g \in G : v(g) = -1\} \subseteq L(1)$. ii) If for each move of Player 1 in state $s$ there exists a move of Player 2 to a state in $L(1)$ then $s$ itself is in $L(1)$. The set $L(2)$ of lost positions for Player 2, is defined analogously. These definitions assume optimal play for both players, where each player chooses the move that maximizes his objective, e.g. Player 2 has to choose a move that forces Player 1 remain in $L(1)$ and so does Player 1 in the analogous case.

Let $R$ be the set of all reachable states, with respect to the initial position and the rules of the game, then $D = R \setminus (L(1) \cup L(2))$ is the set of draw games.

Constructing the above sets $L(1)$, $L(2)$ and $D$ together with their game theoretical value in a backward traversal of the state space of the game is referred to as classification or retrograde analysis.

A general introduction to Two-Person Game Theory is provided by (Rapoport 1966).

### Exploration and Classification Algorithms

We exemplify the algorithmic considerations to compute the set of reachable states and the game theoretical value for large sets of states in the game Tic-Tac-Toe and distinguish the two players by denoting White for Player 1 and Black for Player 2.

Tic-Tac-Toe is a pencil-and-paper two-player game. It is played on a $3 \times 3$-Grid, with alternating ticks of the players. The winner of the game has to complete either a row, a column or a diagonal with his own ticks. Obviously, the size of the state space is bounded by $3^9 = 19,683$ states, since each field is either unmarked or marked with Player 1 or 2. A complete enumeration shows that there is no winning strategy (for either side); the game is a draw.

#### Encoding of States and Transitions

To encode a state $s$ all positions are indexed as Figure 1 visualizes.

```
   1  1  1
   2
   2
```

Figure 1: a) A State in Tic-Tac-Toe b) Labeling of the Board

We devise two predicates: $\text{Occ}(s, i)$ being 1 if position $i$ is occupied, and $\text{Black}(s, i)$ evaluating to 1 if the position $i$, $1 \leq i \leq 9$, is marked by Player 2. This results in a total state encoding length of 18 bits. All final positions in which Player 1 has lost are defined by enumerating all rows, columns and the two diagonals as follows.

\[
\text{WhiteLost}(s) = \\
\{ \text{Occ}(s, 1) \land \text{Occ}(s, 2) \land \text{Occ}(s, 3) \land \\
\text{Black}(s, 1) \land \text{Black}(s, 2) \land \text{Black}(s, 3) \} \lor \\
\{ \text{Occ}(s, 4) \land \text{Occ}(s, 5) \land \text{Occ}(s, 6) \land \\
\text{Black}(s, 4) \land \text{Black}(s, 5) \land \text{Black}(s, 6) \} \lor \\
\{ \text{Occ}(s, 7) \land \text{Occ}(s, 8) \land \text{Occ}(s, 9) \land \\
\text{Black}(s, 7) \land \text{Black}(s, 8) \land \text{Black}(s, 9) \} \lor \\
\{ \text{Occ}(s, 1) \land \text{Occ}(s, 4) \land \text{Occ}(s, 7) \land \\
\text{Black}(s, 1) \land \text{Black}(s, 4) \land \text{Black}(s, 7) \} \lor \\
\{ \text{Occ}(s, 2) \land \text{Occ}(s, 5) \land \text{Occ}(s, 8) \land \\
\text{Black}(s, 2) \land \text{Black}(s, 5) \land \text{Black}(s, 8) \} \lor \\
\{ \text{Occ}(s, 3) \land \text{Occ}(s, 6) \land \text{Occ}(s, 9) \land \\
\text{Black}(s, 3) \land \text{Black}(s, 6) \land \text{Black}(s, 9) \} \lor \\
\{ \text{Occ}(s, 1) \land \text{Occ}(s, 5) \land \text{Occ}(s, 9) \land \\
\text{Black}(s, 1) \land \text{Black}(s, 5) \land \text{Black}(s, 9) \} \lor \\
\{ \text{Occ}(s, 3) \land \text{Occ}(s, 5) \land \text{Occ}(s, 7) \land \\
\text{Black}(s, 3) \land \text{Black}(s, 5) \land \text{Black}(s, 7) \}
\]

The predicate $\text{BlackLost}$ is defined analogously. In order to specify the transition relation we fix a Frame denoting that in the transition from state $s$ to $s'$ besides the move in the actual grid cell $j$ nothing else will be changed.

\[
\text{Frame}(s, s', j) = \\
\{ \text{Occ}(s, j) \land \text{Occ}(s', j) \} \lor \\
\{ \neg \text{Occ}(s, j) \land \neg \text{Occ}(s', j) \} \land \\
\{ \text{Black}(s, j) \land \text{Black}(s', j) \} \lor \\
\{ \neg \text{Black}(s, j) \land \neg \text{Black}(s', j) \}
\]
These predicates are superimposed to express that with respect to board position \( s \) the status of every other cell is preserved.

\[
\text{Frame}(s, s', i) = \bigwedge_{1 \leq i \neq j \leq 9} \text{Frame}(s, s', j)
\]

Now we can express the relation of a black move with origin \( s \) and successor \( s' \). As a precondition we have that one cell \( i \) is not occupied and the effects of the operator are that in state \( s' \) cell \( i \) is occupied and black.

\[
\text{BlackMove}(s, s') = \\
\bigvee_{1 \leq i \leq 9} \neg \text{Occ}(s, i) \land \text{Black}(s', i) \land \\
\text{Occ}(s', i) \land \text{Frame}(s, s', i)
\]

The predicate \( \text{WhiteMove} \) is defined analogously. To devise the encoding of all moves in the transition relation \( \text{Trans}(s, s') \) we address one additional bit \( \text{Move}(s) \) for each state \( s \), denoting the truth of Player 2’s turn, as follows.

\[
\text{Trans}(s, s') = \\
(\neg \text{Move}(s) \land \neg \text{WhiteLost}(s) \land \\
\text{WhiteMove}(s, s') \land \text{Move}(s')) \lor \\
(\text{Move}(s) \land \neg \text{BlackLost}(s) \land \\
\text{BlackMove}(s, s') \land \neg \text{Move}(s'))
\]

There are two cases. If it is Player 2’s turn and if he is not already lost, execute all black moves and continue with a black one.

**Reachability Analysis**

*Symbolic reachability analysis* traverses the entire state space that is reachable from the initial position. Essentially reachability analysis corresponds to a symbolic breadth-first search traversal, which successively takes the set \( \text{From} \) of all positions in the current iteration and applies the transition relation to find the set of all \( \text{New} \) positions in the next iteration. For the iteration to be completed we further need a procedure \( \text{Replace} \) to change the nodes labeling from the association with respect to \( s' \) back to an association with respect to \( s \). Iteration is aborted if no new position is available. The union of all new positions is stored in the set \( \text{Reached} \). All sets are represented by BDDs according to the given encoding. The implementation is depicted in Fig 2.

**Game-Theoretical Classification**

As stated above, two-player games with perfect information are classified iteratively. Therefore, in opposite to reachability analysis the direction of the search process is *backwards*. Fortunately, backward search causes no problem, since the representation of all moves has already been defined as a relation.

Assuming optimal play and starting with all goal situations according to one player – here Black’s lost positions – all previous winning positions – here White’s winning positions – are computed. A position is lost for Player 2 if all moves lead to an intermediate winning position in which white can force a move back to a lost position.

\[
\text{BlackLose}(s) = \\
\text{BlackLost}(s) \lor \exists s' (\text{Trans}(s, s') \Rightarrow \\
(\exists s'' \text{Trans}(s', s'') \land \text{BlackLost}(s'')))
\]

Note that the choice of the operators \( \lor \) for existential quantification and \( \Rightarrow \) for universal quantification are crucial.

**procedure** \( \text{Classify} \)

\[
\text{WhiteWin} \leftarrow \text{false} \]
\[
\text{BlackLose} \leftarrow \text{From} \land \text{BlackLost}(s) \]

**do**

\[
\text{To} \leftarrow \text{Replace}(\text{From}, s, s') \]
\[
\text{To} \leftarrow \exists s' (\text{Trans}(s, s') \land \text{To}(s')) \]
\[
\text{To} \leftarrow \text{To} \land \neg \text{Reach} \land \neg \text{Move}(s) \]
\[
\text{WhiteWin} \leftarrow \text{WhiteWin} \lor \text{To} \]
\[
\text{To} \leftarrow \text{Replace}(\text{To}, s, s') \]
\[
\text{To} \leftarrow \forall s' (\text{Trans}(s, s') \Rightarrow \text{To}(s')) \]
\[
\text{To} \leftarrow \text{To} \land \text{Move}(s) \]
\[
\text{From} \leftarrow \text{New} \leftarrow \text{To} \land \neg \text{BlackLose} \]
\[
\text{BlackLose} \leftarrow \text{BlackLose} \lor \text{New} \]

**while** \( \text{New} \)

**Figure 3:** Determining the set of white winning and black lost positions.

The pseudo-code for symbolic classification is shown in Fig. 3. The algorithm \( \text{Classify} \) starts with the set of all final lost positions for Black, and alternates between the set of positions that in which black (at move) will loose and positions in which white (at move) can win, assuming optimal play. In each iteration each player moves once; corresponding to two quantification in the recursive description above.
One important issue of the pseudo-code is the explicit attachment of the player to move, since this information might not be available in the backward traversal. Furthermore, the computations can be restricted to the set of reachable states through conjunctions with its BDD representation. We summarize this in the following.

1. **Start**\((\text{Config})\): Definition of the initial state for reachability analysis.
2. **WhiteLost**\((\text{Config})\): Final lost positions for white.
3. **BlackLost**\((\text{Config})\): Final lost position for black.
4. **WhiteMove**\((\text{Config}, \text{Config})\): Transition relation for white moves.
5. **BlackMove**\((\text{Config}, \text{Config})\): Transition relation for black moves.

## Complete Exploration

In this section we experiment with implementations to the given interface specification for simple two-player games in C/C++ with J. Lind-Nielsen’s BDD library BuDDy, Release 1.6 (Lind-Nielsen 1999).

### Tic-Tac-Toe

Table 1 depicts the growths of the number of states and the number of BDD nodes in the reachability analysis and the classification algorithm with respect to an increasing search depth \(d\) in Tic-Tac-Toe. The statistics for the BDD structure for BlackLose and WhiteWin are interleaved. Instead of the expected \(9! / 5! 4! = 126\) newly generated states in the last iteration we obtain 78 states. This is due to the fact that we stop state enumeration if a goal has been found.

<table>
<thead>
<tr>
<th>(d)</th>
<th>Reachable (n)</th>
<th>BlackLose (n)</th>
<th>WhiteWin (n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>90</td>
<td>532</td>
</tr>
<tr>
<td>1</td>
<td>37</td>
<td>772</td>
<td>1498</td>
</tr>
<tr>
<td>2</td>
<td>81</td>
<td>1098</td>
<td>1986</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
<td>2330</td>
<td>2350</td>
</tr>
<tr>
<td>4</td>
<td>152</td>
<td>1002</td>
<td>2350</td>
</tr>
<tr>
<td>5</td>
<td>175</td>
<td>1002</td>
<td>1986</td>
</tr>
<tr>
<td>6</td>
<td>344</td>
<td>2350</td>
<td>1986</td>
</tr>
<tr>
<td>7</td>
<td>525</td>
<td>5400</td>
<td>1098</td>
</tr>
<tr>
<td>8</td>
<td>652</td>
<td>5478</td>
<td>1098</td>
</tr>
<tr>
<td>9</td>
<td>656</td>
<td>5478</td>
<td>1098</td>
</tr>
</tbody>
</table>

Table 1: Reachability analysis and classification in the game Tic-Tac-Toe. The numbers of BDD nodes and the sizes of the represented sets of reachable states, white winning and black lost positions are given.

The number of represented states in a BDD corresponds to the number of paths to the one sink. The number of reachable lost positions for black is 626, whereas we have 316 lost positions for white. The number of eventually lost positions for black accumulates to 1098. The number of white winning positions is 1986 and, as expected, the start is not won for white.

### Four Connect

This popular game is played on a vertical rack with 6 rows and 7 columns. Alternatively, black and white pieces are inserted into one pile falling down onto the existing ones. The state space is obviously bounded by \(3^{42}\) and we experiment with encoding of two bits for each of the 42 positions, ordered from bottom to top and right to left. When encoding the height of each column for occupancy \(42 + 7 \cdot 3 = 63\) bits suffice, for a state space of at most \(10^{18}\) states. Tables 2 and 3 depict the results of the reachability analysis. Four Connect has been proven to be a win for the first player in optimal play using a knowledge-based approach (Allis 1998) and minimax-based proof number search (PNS) (Allis 1994), that introduces the third value unknown into the game search tree evaluation. PNS has a working memory requirement linear in the size of the search tree, while \(\alpha\beta\) requires only memory linear to the depth of the tree. Proof-Set Search is a recent improvement to PNS, that trades node explorations for a higher memory consumption (Müller 2001b).

Due to space limitation we have not yet verified Allis’ result. So far, we have only succeeded in full BDD-classifications for 4-Connect up to a \(4 \times 5\) board, which are all draws.

### Hex

Hex is a classical board game invented by the Danish mathematician Hein. The book (Brown 2000) provides a comprehensive report on the history of the game and advanced playing strategies. The board is a hexagonal tiling of \(n\) rows and \(m\) columns. Usually \(m = n\), with 11x11 being the widely accepted standard board size. The rules are simple: players take turns placing a piece of their color on an unoccupied location. The game is won when one player establishes an unbroken chain of ther pieces connection their sides of the board. Since the game can never result in a draw it is easy to prove that the game is won for the first player to move, since otherwise he can adopt the winning strategy of the second player to win the game. Nevertheless the proof is not constructive such that we are still left with the problem to determine the game theoretical value of all intermediate positions. The state space of Hex is bounded by \(3^{n^2}\) as each point may exists in either of three states Empty, White or Black. The current state-of-the-art program Hexy uses a quite unusual approach electrical circuit theory to combine the influence of sub-positions (virtual connections) to larger and larger ones (Anshelevich 2000).

The binary encoding for Hex is similar to the previous example, with two bits per field, since the players are only allowed to set their pieces. Table 4 displays the
results of reachability analysis applied for this encoding. Time consumption in all cases was within 1 minute on a 450 MHz Pentium, and space in the order of 128 MByte was sufficient for exploration.

The challenging question is how to encode the end of the game, i.e. the connection of two opponent sides by the respective color of the players. We proceed as follows. All paths from one node a to a node b of length l are generated in a Divide-and-Conquer style by recursively determining all paths from a to an intermediate node k of length \( \lfloor l/2 \rfloor \) and all paths from k to b of length \( \lfloor l/2 \rfloor \). To avoid re-computations of BDDs we memorize calculated results in a 3-dimensional table \( T[a][b][l] \). With 210 BDD nodes for \( n = 3 \), 424 for \( n = 4 \), and 7206 for \( n = 5 \) the BDD representations of the goal predicates is small, but the enumeration of all paths is very time consuming, such that we have verified correct classification only for these cases.

**Nim**

Nim is another folklore two-player game. We consider the very simple situation of one stack of disks, where each player is allowed to take one, two, or three of them. Note that the n-stack Nim problem reduces to a 1-stack problem by applying combinatorial game theory (Berlekamp, Conway, & Guy 1982). The situation is lost if the resulting vector is zero. The game is lost for the player, who faces the empty stack. The optimal strategy is to enforce a situation with \( 3k + 1 \) disks,
### Endgame Databases in Chess

Several current challenges in single-agent search like Sokoban, Rubik’s Cube and the \((n^2 - 1)\)-Puzzle can be solved with State-of-the-Art implementations and use a common data structure for improving the lower bound estimate on the solution length: The pattern database, in which sub-positions are stored together with their optimal solution length in a relaxed problem space. In two-player games, pattern databases are generated to determine the game-theoretical value for endgames. Pattern databases can be casted as representations of Boolean functions. Instead of computing the value of a function \(f\) for a given input from scratch, a representation of \(f\) is stored in a table. For the input the pattern value can be retrieved in a simple table lookup.

Chess is one of the oldest games known to mankind. The book (Heinz 2000) provides a computer-chess primer and new results of computer-chess. Chess has advanced from the *drosophila* of AI to one of its main successes, resulting in the defeat of the human-world champion in a tournament match. Some combinatorial chess problems like the number of \(33,439,123,484,294\) complete Knight’s tours have already been solved with BDDs (Löbbing & Wegener 1996). Due to the complexity of chess we are restricted to the construction of endgame databases. Not only the size of the chess board but also the numerous available moves lead to intractable large database for all states and transitions. Early work on endgame databases is surveyed in (van den Herik & Herschberg 1986). Nowadays, Edward’s table-bases and Thomson’s databases are most important to the chess community. The major compression schemes for positions without pawns (since these situations contain only totally reversible moves) use symmetries along the diagonal, horizontal and vertical middle axes of the chess board. Since the 3-fold symmetry allows for the confinement of one piece in a triangular region of 10 squares, the obtained reduction ratio is about \((64 - 10)/64 = 84.38\%\).

### Example Setting

As a first example we consider a board with two opposing Kings and one white Queen. In the encoding of the moves for the kings we restrict successor generation to squares not threatened by another figure. Movements of the queen are more complicated. The Queen can freely move on rows, columns and diagonals as long as no other figure is present. Removal is only allowed if a figure of the opposite color is encountered at the destination square. However, captures in this simple problem instance simply terminates the game since the remaining game is either a draw (two opposing Kings) or a definite win for white. Opposite to Tic-Tac-Toe in each move two squares are changed for each move. Therefore the Frame has to be adequately enlarged with this further parameter.

The simple encoding with three bits for each square denoting occupancy, color and figure type, together with one bit for the player to move, yields 193 bits in total, a number for which symbolic exploration turns out to be intractable. The BDD sizes simply exhaust main memory. A more suitable encoding of board positions is a binary representation of the occupied squares for the two kings and the queen. Since six bits per figure suffice to encode 64 squares this gives a total of 19 bits.

In a specialized implementation of the example problem the chess configuration *ChessConfig* is realized by three simple predicates: \(qw(i,j)\) for row \(i\) and column \(j\) returns a BDD, evaluating to 1, if and only if the white queen is positioned at \((i,j)\). The procedures \(kw(i,j)\) and \(ks(i,j)\) yield analogous BDD representations for the black and white king, respectively. The constructor includes a switch for the different variable sets in the situations prior and after move commitment. The class *Chess* then implements all above methods, while we have added two methods *BlackThreatened* and *WhiteThreatened* that indicate, if a given square is available for the player black and white, respectively.

---

**Table 4: Final results of the reachability analysis for the game Hex scaled with parameter \(n\) denoting an \(n \times n\) board. The final depth of the analysis \(l\), the BDD-size of the transition relation and the numbers of BDD nodes and represented states for the complete exploration are presented.**

<table>
<thead>
<tr>
<th>(n)</th>
<th>(l)</th>
<th>Transition Relation (n)</th>
<th>Reachable (n)</th>
<th>(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>171</td>
<td>144</td>
<td>6,046</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>311</td>
<td>421</td>
<td>1.01e+07</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>491</td>
<td>1,000</td>
<td>1.61e+11</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>711</td>
<td>2,034</td>
<td>2.40e+16</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>971</td>
<td>3,724</td>
<td>3.30e+22</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>1,271</td>
<td>6,304</td>
<td>4.15e+29</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>1,611</td>
<td>10,044</td>
<td>4.78e+37</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1,991</td>
<td>15,250</td>
<td>5.00e+46</td>
</tr>
<tr>
<td>11</td>
<td>121</td>
<td>2,411</td>
<td>22,264</td>
<td>6.87e+10</td>
</tr>
<tr>
<td>12</td>
<td>144</td>
<td>2,871</td>
<td>31,464</td>
<td>8.01e+13</td>
</tr>
</tbody>
</table>

**Table 5: The game Nim with an either unary or binary encoding of natural numbers. The numbers of BDD nodes and represented states for the entire set of reachable states are provided. In the binary encoding equivalent states have been merged.**

<table>
<thead>
<tr>
<th>(n)</th>
<th>(s)</th>
<th>(n)</th>
<th>(s)</th>
<th>(n)</th>
<th>(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>11</td>
<td>58</td>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>2,037</td>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>31</td>
<td>65,520</td>
<td>6</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>41</td>
<td>2.99e+06</td>
<td>8</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>51</td>
<td>6.71e+07</td>
<td>7</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>61</td>
<td>2.14e+09</td>
<td>7</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>71</td>
<td>6.87e+10</td>
<td>8</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>81</td>
<td>2.14e+09</td>
<td>10</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
General Setting
In a more general case of arbitrary figures on the board exploration is based on the BDD representation of the figures’ positions $At(i,j)$, $1 \leq i,j \leq 8$. This implementation scans the endgame database description and the initial state for reachability analysis in the command line.

We distinguish between white and black occupancy of squares, since according to the color of the pieces the target position might lead to a capture. For Queens, Bishops and Rooks all intermediate positions within one move are blocked. Therefore, to specify a move beside the pre-calculated Frame we pre-compute two vectors EmptyWhite and EmptyBlack representing the emptiness of each square. EmptyWhite (EmptyBlack) w.r.t. $(i,j)$ evaluates to 1, if no white (black) figure is currently located at position $(i,j)$. For example, all Knight jumps in the direction up-up-left are characterized as follows.

\[
\bigvee_{2<i\leq8,1<j\leq8} At(s,i,j) \land At(s',i-2,j-1) \land \\
\text{EmptyWhite}(s,i-2,j-1) \land \text{Frame}(s,s')
\]

A capture is found by querying variables equality for each pair of figures on the board. This predicate simply assigns that the binary value of the board positions of different figures coincide. Figures that have been captured are placed onto an extension of the board.

In order to specify the Check and Mate positions, we apply the classification algorithm for one and two plies (half-move). A position is Mate if even in perfect play the foreign king is definitely captured in one move and a position is Check if the king can be taken assuming a void move of the opponent. Therefore, a Check-Mate is the conjunct of the former two predicates. Table 6 gives our preliminary results. For these simple endgame studies the game can be terminated if one of the player takes a figure of the opponent. In general we encounter a database entry of a smaller game.

<table>
<thead>
<tr>
<th></th>
<th>Check-Mate</th>
<th>Reachable</th>
</tr>
</thead>
<tbody>
<tr>
<td>KKR</td>
<td>385</td>
<td>75</td>
</tr>
<tr>
<td>KRRK</td>
<td>8,829</td>
<td>794</td>
</tr>
<tr>
<td>KRNK</td>
<td>2,395</td>
<td>954</td>
</tr>
<tr>
<td>KNNK</td>
<td>804</td>
<td>1,050</td>
</tr>
<tr>
<td>KNBR</td>
<td>519</td>
<td>1,148</td>
</tr>
<tr>
<td>KBRK</td>
<td>1,295</td>
<td>641</td>
</tr>
</tbody>
</table>

Table 6: Endgame databases in Chess according to different situations. The letter N abbreviates Knight, K denotes King Q Queen, R Rook, B Bishop.

Reachability and classification took less than 30 seconds on a 450 MHz Pentium. The number of represented states in the final set of reachable states exceeds the BDD representation by magnitudes. In classifying the example KBRK, the set for states eventually won for the first player has had 312932 elements with a BDD representation of 15,460 nodes, which corresponds to a saving of about 95%; the larger the number of states, the better the gain by BDD representation.

In all cases backward analysis is more time consuming than forward exploration. This observation not necessarily reflects BDD sizes but to the number of subfunctions encountered in existential and universal quantification. Moreover, the number of subproblems is related to the number of represented states. Therefore, backward iterations corresponds to a sizable amount of work if the number of represented states in the goal predicate is large.

After a successful classification the optimal strategy for Player 1 can be obtained by memorizing the sets of states WhiteWins in each iteration. If Player 2 has performed his move a simple conjunction of the available successors with the WhiteWins gives the next winning position.

Endgame database queries can be decided in time linear to the encoding length. Therefore, BDD endgame databases can compete with hashing schemes to query the game-theoretical value in ordinary endgame databases.

The symbolic representation of a BDD $B$ transforms to an explicit one by extracting and deleting one represented state after another. A satisfying path $p$ in the $B$ is extracted as a BDD $P$ and $B$ is updated to $B \wedge \neg P$ until $B$ represents false. One application is a print routine for the PDB.

A Framework for Multi-Player Games
Let $Q$ be the set of all states and for $G = \{s \in Q \mid \text{goal}(s)\}$ we define an evaluation function $v : G \rightarrow \{0,1,\ldots,k\}$, with $k$ being the number of players and the value $i > 0$ corresponds to a definite win for Player $i$.

As in the two-player scenario this function is extended to $\hat{v}: Q \rightarrow \{0,1,\ldots,k\}$ by induction. The set of winning positions for player $i$, i.e. all positions $s \in Q$ with $\hat{v}(s) = i$, is calculated as follows.

\[
\text{PlayerWin}(i,s) = \\
\text{PlayerWon}(i,s) \lor \\
\forall_{s^{(1)}} (\text{Trans}(s^{(0)},s^{(1)}) \Rightarrow \\
(\forall_{s^{(2)}} \text{Trans}(s^{(1)},s^{(2)}) \Rightarrow \\
\ldots \\
(\forall_{s^{(k)}} \text{Trans}(s^{(k-1)},s^{(k)}) \Rightarrow \\
(\exists_{s^{(k+1)}} \text{Trans}(s^{(k)},s^{(k+1)}) \land \\
\text{PlayerWin}(i,s^{(k+1)}) \ldots))
\]

Other Domains and Their Encoding
Due to the depth and diversity of the research in the area of game-playing (Schaeffer 2000) we can only indi-
order bounding (Müller 2001a) to propagate relative of tractable size. In these sub-searches BDD databases some board situations are split into a sum of local games method based on combinatorial game theory in which in some endgame situations uses a Divide-and-Conquer tant approach (Müller 1995) with exponential savings has been addressed by different strategies. One import-

nite too large for a complete symbolic exploration. Go The resulting binary encodings of over 500 bits are defi-
mens might lead to a more complicated transition relations.

In its time the retired human-computer world-champion Checkers program Chinnock has performed its intense endgame database computations (up the 7 and 8 pieces) on various high-end computers in the US and Canada. All checker positions involving 8 or fewer pieces on the board, result in a total of 443,748,401,247 positions. In a simple encoding Checkers requires 32 · \lfloor \log_5 32 \rfloor = 96 bits. As above an improved encoding might get closer to the reasonable bound of \lfloor \log_5 32 \rfloor = 75 bits. Although this seems tractable for a BDD engineer, experiments with the Fifteen-Puzzle in a concise encoding of 64 bits has shown that reachability analysis easily exhausts 500 MByte of memory with symbolic breadth first search. Therefore, for the appli-

cation of presented approach in Checkers we suggest to increase the bound on endgame computations.

The size of the search space in Go (19 × 19 variant) has been estimated at 10^{170} positions (3^{19·19} ≈ 10^{172}), and is probably the biggest of all popular board games. The resulting binary encodings of over 500 bits are defi-
nitely too large for a complete symbolic exploration. Go has been addressed by different strategies. One important approach (Müller 1995) with exponential savings in some endgame situations uses a Divide-and-Conquer method based on combinatorial game theory in which some board situations are split into a sum of local games of tractable size. In these sub-searches BDD databases might be advantageous. The general strategy of partial order bounding (Müller 2001a) to propagate relative evaluations in the tree has been shown to be effective in Go endgames; it applies to all minimax search algo-
rithms such as αβ and PNS.

One multi-player game is Halma/Chinese Checkers with its star-like game board introduced at the end of the 19th century. It can be played with up to 6 players. The goal of the game is to move all own pieces to the opposing side of the board by sliding single pieces to adjacent places or jumping over adjacent pieces if the destination is free. Chaining of jumps is allowed and reveals the tactics of the game. One obvious encoding considers 3 bits for each of the 121 board positions to denote the occupancy of the pieces such that this game is likely to be too complex to be solved with current BDD technology.

\section{Conclusion and Outlook}

To the authors’ knowledge, this paper is one of the first manuscripts on using BDDs to classify two-player games. The only other work we are aware of is (Bal-
damus et al. 2002), that applies a Model Checker to solve American Checkers problems.

In the experiments we highlighted possible memory savings for the complete exploration in two simple problems Nim and Tic-Tac-Toe and a medium-size problem Four Connect. With Hex and Chess we gave examples, where the binary encodings of the moves and the end-
ing is not trivial. The results are preliminary, but lead to drastic savings in the considered problem spectrum. The gap often corresponds to several magnitudes. We generalized the approach to the the multi-player scen-
ario which especially in card-games attracts several researchers nowadays (cf. (Ginsberg 1999) for an ex-

ample).

The results are preliminary. In large instances to Hex and in Four Connect we have not yet determined the game-theoretical value of all states for larger problem instances with the symbolic traversal of the search space. Moreover, the domain of chess gives only rather trivial results (King + 1 or 2 pieces vs. King), which state spaces have already fully been explored by many computer-chess programmers. Last but not least, the indicated application of the methods in Nine-Men-Morris, Checkers and Halma is quite speculative and yet not been implemented. However, these limitation are not necessarily problem-inherent, and symbolic repre-
sentation and exploration is promising to enrich the portfolio of game playing programs. Moreover, the simplic-
ity and generality of the approach can serve as an inter-
face for specifying simple two-player games with op-
timal play. The next two options to improve the perfor-

mane are static or dynamic variable ordering schemes, which usually have a significant influence on the space and time complexity, and partitioning techniques of the search space to bypass bottle-necks in the symbolic exploration process. As in the case of chess, game playing problem instances are often redundant with respect to different automorphisms to be exhibited by combined reduction schemes and advanced data structures.

Since reachability analysis is in practice easier and faster than classification, BDDs might support the eval-
uation of successors in active play as follows. Take the successor state and evaluate the game all the way down to the end and draw statistics on how often the goal situation is met for both players. The successor with the best score for one player is searched first. Enum-
oration bases on recent progress in game playing, since its successful variant is random sampling that has been applied in Bridge, where Monte Carlo Sampling determines the hands of the opponents (Ginsberg 1999) and to Backgammon, where a Roll-Out reveals the current strength of the game (Tesauro 1995).

Another reason why counting might be an advan-
tage to αβ pruning in min-max search is that this
algorithm is sensible to the number of leaf nodes responsible for the root evaluation even though alpha-beta tends to hide errors at leaf nodes. This problem has been addressed by conspiracy number search (CNS) (McAllester 1988). The basic idea of CNS is to search the tree in a manner that at least $c > 1$ leaf values have to change in order to change the root one. CNS has been successfully applied to chess by (Schaeffer 1989) and (Lorenz 2000).

As a final side remark, note that two player exploration joins many features with adversarial universal planning for multi-agent domains in which a set of uncontrollable agents may be adversarial to the planner (Jensen, Veloso, & Bowling 2001).

Acknowledgment  The author would like to thank DFG for the support in the projects Ot 64/13-2 and Ed 74/2-1.

References


