ECCC
TR95-047

FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

The Number of Knight's Tours Equals 33,439,123,484,294 — Counting with Binary Decision Diagrams

Martin Löbbing[*] and Ingo Wegener[*]

FB Informatik, LS II, Univ. Dortmund, 44221 Dortmund, Germany
e-mail: `loebbing/wegener@ls2.informatik.uni-dortmund.de`

Abstract

An increasing number of results in graph theory, combinatorics and theoretical computer science is obtained with the help of computers, e. g. the proof of the Four Colours Theorem or the computation of certain Ramsey numbers. Binary decision diagrams, known as tools in hardware verification and computer-aided design, are used here for the solution of counting and other combinatorial problems. The basic results about the appropriate variants of binary decision diagrams are presented. In order to prove the usefulness of the approach the number of knight's tours on an 8×8 chessboard is computed for the first time as well as the number of cycle coverings. Moreover, bounds on the asymptotic number of knight's tours are discussed.

[*]Supported in part by DFG grant We 1066/7–3.

1. INTRODUCTION

Although everybody knows the rules of chess, we define formally the knight's graph for $n \times n$ chessboards. It consists of n^2 nodes (i, j), $1 \leq i, j \leq n$, representing the squares of the chessboard. Two nodes (i, j) and (i', j') are connected by an edge iff a knight can move in one step from (i, j) to (i', j'), i. e. $|i - i'| = 1$ and $|j - j'| = 2$ or $|i - i'| = 2$ and $|j - j'| = 1$. A knight's tour is a Hamiltonian cycle on the knight's graph. A well known relaxation of the traveling salesman problem, i. e. the computation of a minimal cost Hamiltonian cycle, is the so-called assignment problem (see Lawler, Lenstra, Rinnooy Kan, and Shmyos (1985)), where the graph has to be covered by disjoint cycles, not necessarily one cycle.

Why should one compute the number of knight's tours and cycle coverings on 8×8 chessboards? Who is interested in these numbers? Actually, perhaps nobody is interested in the concrete numbers. Our motivation is based on a more general argument. Binary decision diagrams are representations of Boolean functions with many applications in hardware verification and computer-aided design (for surveys see Bryant (1992) and Wegener (1994)). We believe that binary decision diagrams also have a lot of applications for problems in combinatorics, graph theory and theoretical computer science. This claim has to be justified by general arguments and the solution of an open problem. There are many reasons why we have chosen the problem of counting tours on the knight's graph.

- The problem is one of the two most famous combinatorial chess problems, the other one is the n-queens problem.
- Combinatorial chess problems are known to almost everybody and not only to a specialized group of researchers.
- The knight's problem has a long history. Famous mathematicians like Euler, Legendre, and Vandermonde (see e. g. Rouse Ball and Coxeter (1987)) have worked on this problem.
- If $T(n)$ denotes the number of undirected knight's tours on an $n \times n$ chessboard, nobody expects the existence of a "short" formula for $T(n)$ which can be "easily" evaluated. (It is known that $T(n) > 0$ iff n is even and $n \geq 6$.)
- It is in the nature of the problem that we do not expect the existence of a "short" recursion relation for $T(n)$ which can be easily evaluated for "small" n.
- Chess problems have the property that the result for one special n is more interesting than asymptotic results. The simple reason is that the natural chessboard has size 8×8.

So we compute the numbers $T(8)$ and $C(8)$ (number of directed cycle coverings on the 8×8 chessboard) in order to prove the usefulness of binary decision diagrams for concrete counting problems. These numbers were unknown before. We shall argue why we obtain these numbers with binary decision diagrams much faster than without binary decision diagrams.

Our approach is part of the development to solve more and more problems directly with the help of computers. We only sketch the history of this development. Later we shall see that binary decision diagrams include many properties of other approaches.

In theoretical computer science as well as in mathematics problems are usually solved by the (more or less) formal proof of a theorem. Computers are used to establish conjectures or to support the solution of technical problems (Maple or Mathematica). In the recent years an increasing number of results in graph theory, combinatorics and theoretical computer science is proved directly with the help of computers. The most famous example is the theorem that every planar graph is four colorable (Appel and Haken (1977) and Appel, Haken, and Koch (1977)). Since we consider chess problems, we also mention the description of all pairs (s, t) of squares on $n \times n$ chessboards such that the knight has a Hamiltonian path from s to t (Conrad, Hindrichs, Morsy, and Wegener (1994)). Computers are useful for these results, since the number of basic cases which have to be solved is too large to be handled without computers.

Other problems are completely solved with computers. A typical example is the computation of Ramsey numbers (for a survey see Graham and Rödl (1987)). The Ramsey number $R(k, l, s)$ is defined to be the least n such that, in any coloring with two colors of the s-subsets of a set of n elements, there is a k-subset all of whose s-subsets have the first color or there is an l-subset all of whose s-subsets have the second color. These problems can be interpreted as hypergraph problems on hypergraphs whose edges contain s nodes. Therefore, $s = 2$ is the graph case and $R(k, l) = R(k, l, 2)$ are known as Ramsey numbers for graphs. The problem has some similarity with our chess problem. Nobody expects a short exact formula, asymptotic results are not very tight and one is interested in concrete Ramsey numbers. E.g., $R(5, 5)$ is unknown. The computation of $R(5, 5)$ is possible in finite time by case inspection. But we shall never live to see the day where $R(5, 5)$ is computed by exhaustive search. Another Ramsey number has been determined with computers. McKay and Radziszowski (1991) have proved that $R(4, 4, 3) = 13$. The computer is used for the following purposes.

- Solution of linear programming problems.
- Solution of integer linear programming problems.
- Early recognition of isomorphic cases.
- Early recognition of subproblems without solution.

In order to underline the generality of our approach we do not only solve the knight's problems but we shall also discuss how a similar approach may be used for Ramsey problems.

In Section 2 the general approach to solve combinatorial and counting problems with binary decision diagrams (BDDs) is presented. In Section 3 the different BDD variants used later are introduced. Structural properties are discussed and efficient algorithms for the manipulation of BDDs are presented.

In Section 4 the exact number of coverings of the 8×8 knight's graph is computed and in Section 5 the exact number of knight's tours is computed. It is also discussed whether one can trust these results obtained with the computer. Implications to asymptotic results are discussed in Section 6.

2. HOW TO SOLVE COMBINATORIAL PROBLEMS WITH BINARY DECISION DIAGRAMS

We describe our general approach to solve combinatorial problems with binary decision diagrams (BDDs). As example we mainly consider the counting problems on the knight's graph which we solve later. In order to underline the generality of our approach we also describe how one may attack a completely different problem, the computation or estimation of the Ramsey number $R(5,5)$.

In this section we treat BDDs as abstract data type for the representation of Boolean functions and collect the operations for which we need efficient algorithms. The definition of several BDD variants together with algorithms for the manipulation of these BDDs follow in the next section.

We only discuss finite problems. It is quite natural to model them as Boolean functions. The knight's graph G_n for $n \times n$ chessboards is a fixed graph whose nodes have degree 2, 3, 4, 6 or 8. Cycle coverings and knight's tours are subgraphs where each node has degree 2. Considering the directed version each node has indegree 1 and outdegree 1. For a node v with degree d we reserve $\lceil \log d \rceil$ Boolean variables. E.g., for a node with degree 3 we have 2 Boolean variables with 4 possible values. One of the values is declared as illegal, the others describe the possible successors. The Boolean function $\mathrm{COV}(x)$ computes 1 iff the Boolean vector x describes a cycle covering (in particular a legal successor for each node). The Boolean function $\mathrm{TOUR}(x)$ computes 1 iff x describes a knight's tour. The number of directed cycle coverings is equal to the number of satisfying inputs of COV and the number of directed knight's tours is equal to the number of satisfying inputs of TOUR (the number of undirected knight's tours is half the number of directed ones). Already the satisfiability test problem is NP-complete for Boolean functions represented by circuits. We shall consider BDD variants where it is easy to count exactly the number of satisfying inputs of the represented function. For the Ramsey problem we consider $\binom{n}{2}$ Boolean variables x_{ij}, $1 \le i < j \le n$, describing an undirected graph $G(x)$ on n nodes. The Boolean function $R_5^n(x)$ equals 1 iff $G(x)$ contains a 5-clique or a 5-anticlique. The Ramsey number $R(5,5)$ is the least n such that $\overline{R_5^n}$ is not satisfiable. Counting the satisfying inputs for $\overline{R_5^n}$ and $n < R(5,5)$ gives the number of $(5,5)$-good graphs (for the importance of these numbers see McKay and Radziszowski (1994)).

In general we have the formal description of a Boolean function. Our approach can only be successful, if the description is short, i.e. has polynomial length. This is the case in our examples. It is easy to test whether a vector x describes a cycle covering (indegree 1 and outdegree 1) or even a Hamiltonian cycle (indegree 1, outdegree 1 and the subgraph is strongly connected). This leads to circuits of small polynomial size. The reason is that the circuit does not decide for arbitrary graphs whether they contain a Hamiltonian cycle but only for a specific graph whether some choice of edges builds a Hamiltonian cycle. For the Ramsey problem it is sufficient to check whether a given graph contains a 5-clique or a 5-anticlique. This can be checked by a circuit of size $O(n^5)$.

The circuit description and many other succinct formal descriptions do not allow efficient satisfiability checks and counts. Our approach is to try to translate the given description into a representation by a BDD variant for which satisfiability checks and counts can be performed efficiently. From the NP-completeness theory it is obvious that not every small circuit will lead to small BDDs. But we can hope to solve some problems. The transformation from a formal description to a BDD is done step by step, for a circuit gate by gate. The variables considered as Boolean functions will have small BDDs. Then we need an efficient synthesis algorithm to obtain for a binary Boolean operator $\otimes$ a representation for $f = g \otimes h$ from representations for g and h. Even if the representation size increases in one synthesis step only slowly it may happen that it increases too much in a lot of synthesis steps. This is the reason why our approach may fail.

The following operations have to be supported by BDD variants we use.

- Satisfiability test. Given some representation G for f. Check whether $f \equiv 0$.

- Satisfiability count. Given some representation G for f. Count the number of inputs a such that $f(a) = 1$.

- Synthesis. Given some representations G_g and G_h for g and h and a binary Boolean operator $\otimes$. Compute some representation G_f for $f = g \otimes h$.

Some other features are highly desirable. A sequence of synthesis steps may lead to a large representation of a function which also has short representations. Hence, we should be able to minimize the representation size. For many BDD variants the representation of minimal size will be unique and then called the reduced representation. Efficient reduction algorithms are very useful.

If we want to solve subproblems, we are faced with the problem that some vector of variables $(x_{i(1)}, \dots, x_{i(k)})$ should be equal to the vector $(a_{i(1)}, \dots, a_{i(k)}) \in \{0,1\}^k$. For this replacement by constants we ask for efficient algorithms as well as for the corresponding problem to ensure that $(x_{i(1)}, \dots, x_{i(k)}) \neq (a_{i(1)}, \dots, a_{i(k)})$.

3. BINARY DECISION DIAGRAMS: VARIANTS AND ALGORITHMS

Binary decision diagrams (BDDs) are known in complexity theory as branching programs.

Definition 1: A BDD is a directed acyclic graph with two sinks labeled by the Boolean constants and inner nodes labeled by Boolean variables. Each inner node has an outgoing 0-edge and an outgoing 1-edge. Each node v represents a Boolean function f_v, where $f_v(a)$ is computed in the following way. Start at v and at an inner node with label x_i follow the a_i-edge. Then $f_v(a)$ is equal to the label of the sink finally reached.

General BDDs are not adequate for our purposes, since the satisfiability problems are too hard. This is caused by the fact that graph theoretical paths may contain 1-edges leaving x_i-nodes as well as 0-edges leaving x_i-nodes. Hence, restricted variants of BDDs are used in applications. Among the many variants we present only those which we have used.

Definition 2: i) An FBDD (free BDD or read-once branching program) is a BDD where each path contains for each variable x_i at most one x_i-node.

ii) An OBDD (ordered BDD) for the variable ordering $x_1, \ldots, x_n$ is an FBDD where each edge leaving an x_i-node leads to a sink or an x_j-node with $j > i$.

The satisfiability problems can be solved in linear time for FBDDs and OBDDs, since each graph theoretical path is also a computation path for at least one input.

Let *start* be an imaginary node labeled by x_0 and connected to all other nodes. We interpret an edge from an x_i-node to an x_j-node (the sinks are viewed as x_{n+1}-nodes) as 2^{j-i-1} edges (for the possible values of $x_{i+1}, \ldots, x_{j-1}$). Then the function represented at the x_i-node v has as many satisfying inputs as there are paths from *start* via v to the 1-sink. Let v_0 and v_1 be the successors of v, n_0 and n_1 the number of paths from v_0 and v_1 to the 1-sink and m_0 and m_1 the multiplicity of the edge (v, v_0) resp. (v, v_1). Then the number of paths from v to the 1-sink equals $m_0 n_0 + m_1 n_1$ and can be computed in constant time.

The synthesis problem for FBDDs causes problems. We know of two functions of linear FBDD size whose conjunction has exponential FBDD size. Even if the resulting FBDD size is small there is no algorithm guaranteeing a good run time for the synthesis. This is the reason why OBDDs introduced by Bryant (1986) are most often used in applications. All operations can be performed efficiently. For the synthesis problem we remark that OBDDs are finite automata, where it is allowed to omit letters. Synthesis algorithms for finite automata are well-known (Hopcroft and Ullman (1979)).

OBDDs are acyclic. This property allows Bryant (1986) to include the reduction process into the synthesis algorithm. This is essential, since applications fail much more often because of lack of space than because of lack of time. The necessary space is the space for the given OBDDs G_g and G_h, the constructed reduced OBDD G_f and a dictionary for all pairs (v_g, v_h) of nodes in G_g and G_h which have been visited in the synthesis process. The synthesis is only time efficient, if all the data is contained in the main storage. The problem is the dictionary. In applications this problem is solved by hashing strategies using a cache storage of limited space. This is usually quite efficient, although the worst case can become exponential (Brace, Rudell, and Bryant (1990)).

The replacement of variables is easy. If $(x_{i(1)}, \ldots, x_{i(k)}) = (a_{i(1)}, \ldots, a_{i(k)})$, it is sufficient to replace all $x_{i(j)}$-nodes by their $a_{i(j)}$-successor. The problem to ensure that $(x_{i(1)}, \ldots, x_{i(k)}) \neq (a_{i(1)}, \ldots, a_{i(k)})$ is a little bit more complicated. It can be seen as conjunction of the given function and the negation of the monom $(x_{i(1)}^{a_{i(1)}} \wedge \ldots \wedge x_{i(k)}^{a_{i(k)}})$. In some of our applications (see Section 4) we need such "negative replacement" for variable vectors concerning one square of the chessboard. Then we have used ordered multi decision diagrams (MDDs), where we have one variable x_s with d possible values for each square s with d neighbors. Then $x_j \neq i$ can be performed by a replacement of the i-successor of each x_s-node by the 0-sink.

Minato (1993) has introduced zero-suppressed BDDs (ZBDDs) which have some advantages for functions, where most of the inputs are mapped to 0 (like in our applications).

Definition 3: A ZBDD has the same syntax (graph theoretical structure) as an OBDD. For the computation of $f_v(a)$ the same path is considered. If the 1-sink is reached, $f_v(a) = 1$ only if the path contains x_i-nodes for all i such that $a_i = 1$.

Minato (1993) has described some algorithms for ZBDDs and we have produced a whole ZBDD package with algorithms for all relevant operations. The main result is that the satisfiability problems and the synthesis problem are as efficient as for OBDDs. The same holds for the replacement of a variable by the constant 1. The replacement of a variable by the constant 0 may cause an increasement of the size by a factor of 3/2. The increasement is limited by a factor of n, if we replace several variables by 0. The ZBDD size and the OBDD size are related (in both directions) by a factor of n. Theoretically, this is only a polynomial factor but the factor may be relevant in applications.

All our considerations have been made for a fixed variable ordering. The choice of an adequate variable ordering is a most important issue. This problem is known to be hard (Bollig, Savicky, and Wegener (1994)) but a lot of good heuristics exist (Panda and Somenzi (1995)).

OBDDs are a quite restricted model and it is easy to see that intuitively simple functions have OBDDs of exponential size with respect to every variable ordering. Why are OBDDs nevertheless so successful? The reason is that algorithms for OBDD manipulation perform the reduction automatically. Hence, we work with OBDDs of minimal size. This includes the automatic solution of two problems mentioned in Section 1, the early recognition of isomorphic cases and of subproblems without solution. If two cases are isomorphic, they are described by identical Boolean functions which are represented in OBDDs of minimal size by the same node. If a subproblem has no solution, it is described by the constant 0. Hence, it is represented in OBDDs of minimal size by the 0-sink. Two main problems which we are faced with in search algorithms (exhaustive search, backtracking algorithms) become obsolete, if we work with OBDDs! We remark that another BDD variant, edge-valued BDDs (EVBDDs) introduced by Lai, Pedram, and Vrudhula (1994), supports algorithms for integer linear programming.

After all one may ask which combinatorial problems have already been solved with BDD techniques. There have been only two attempts. Semba and Yajima (1994) model a lot of graph theoretical problems as Boolean problems. But they solve no open problem. Minato (1994) attacks the n-queens problem with ZBDDs. He obtains solutions for $n \leq 13$ but the size of his ZBDDs grows "almost proportional with the number of solutions". Hence, one can expect for $n = 16$ a ZBDD with more than $40 \cdot 10^6$ nodes which is not manageable. Backtracking algorithms are much better for this problem. The exact solution of the 16-queens problems can be obtained in short time. This shows that OBDDs are not a "magic" tool. Our applications prove that one needs a lot for a success. One has to choose an appropriate modeling of the problem, an appropriate BDD variant, an appropriate variable ordering and, this is perhaps the most important issue, one has to combine BDD-techniques with other techniques for the design of efficient algorithms.

Hence, we do not claim to present an automatic machinery to attack all kinds of combinatorial problems.

4. CYCLE COVERINGS OF THE KNIGHT'S GRAPH

Theorem 1: The number of cycle coverings of the directed knight's graph for 8×8 chessboards is equal to α^2, where $\alpha = 2,849,759,680$, i.e. the number equals $8,121,130,233,568,102,400 \approx 8\cdot 10^{18}$.

Proof: This is an example of a quite direct application of BDD techniques. The knight's graph is bipartite, the knight can move only from a white square to a black one and vice versa. Each cycle covering consists of 32 node disjoint edges starting at the white squares and reaching the black squares and 32 node disjoint edges from the black to the white squares. Each choice of such edges also leads to a cycle covering. Moreover, the knight's graph is symmetric with respect to white and black squares. Hence, the number of cycle coverings equals α^2, if α is the number of sets of 32 node disjoint edges from white to black squares.

In order to compute α we use ZBDDs. Each white square with d neighbors gets $\lceil \log d \rceil$ Boolean variables which describe the chosen successor. Altogether 78 Boolean variables are sufficient. We design a circuit to check whether the Boolean variables describe 32 node disjoint edges. It is sufficient to test that each black square is reached by a chosen edge. The circuit is transformed with our synthesis algorithm into a ZBDD of 406,660 nodes (OBDDs have 598,472 nodes). The chosen variable ordering tests the variables for each square blockwise and the white squares are ordered rowwise from left to right. Finally, α is computed by counting the satisfying inputs for the constructed ZBDD. Some amount of storage space is necessary for algorithms on BDDs. A SUN 670/140 with 128 MB storage space is able to compute α in 6.5 minutes. $\square$

The determination of the number of cycle coverings for the 8×8 chessboard seems to be the first combinatorial result which has been obtained for the first time with BDD techniques. We have to admit that α can also be computed with backtracking algorithms. Our most clever implementation took more than 30 days. Hence, we strongly believe that BDD techniques are much faster for the computation of α than any other known technique.

Can we trust in Theorem 1? Software may be faulty and even the hardware may be faulty. But we have obtained the same number with three independent approaches: ZBDDs, OBDDs and backtracking.

Let us assume that we do not recognize that the knight's graph is bipartite and that it is sufficient to compute α. The resulting ZBDD for the 156 Boolean variables describing the 64 chosen edges and for the function checking whether the variables describe a cycle covering has twice the number of inner nodes of the ZBDD for the computation of α. This follows from a simple observation. If $g(x)$ and $h(y)$ are defined on disjoint sets of variables, the OBDD/ZBDD size of $f(x,y) = g(x) \wedge h(y)$ (number of inner nodes) is the sum of the OBDD/ZBDD sizes of g and h, if all x-variables are tested before all y-variables or vice versa. Hence, a direct computation of α^2 is possible in a reasonable amount of time and space. One may imagine the problems with backtracking algorithms.

We can play with our techniques. We can fix the edges leaving some squares and then prevent with MDD techniques that these squares lie on cycles of length 2. If we sum the numbers of these restricted cycle coverings for all possibilities to fix the edges for the chosen squares, we obtain upper bounds on the number of knight's tours. If we choose 16 squares of degree 2, 3 and 4, we obtain an upper bound of less than $0.7 \cdot 10^{15}$ for the number of undirected knight's tours. In the next section we present methods for the exact determination of the number of knight's tours.

5. KNIGHT'S TOURS ON AN 8×8 CHESSBOARD

It is known that a knight's tour exists on an $n \times n$ chessboard, if and only if $n \geq 6$ and n is even (Conrad, Hindrichs, Morsy, and Wegener (1994)). For $n = 6$ it is easy to enumerate the 9,862 undirected tours, even with backtracking algorithms. But for the natural 8×8 chessboard the number of deadlocks is so large that backtracking leads in a reasonable amount of time only to the enumeration of about 10^9 knight's tours. Heuristics like the one by Warnsdorff (1823) only speedup the construction of the first knight's tours. Also pure BDD techniques need too much resources. We see no possibility to construct some BDD variant (with efficient satisfiability count algorithms) for the representation of knight's tours in the nowadays available storage space. Nevertheless, we determine the number of knight's tours on an 8×8 chessboard.

Theorem 2: The number of undirected knight's tours on an 8×8 chessboard is equal to

$$33{,}439{,}123{,}484{,}294.$$

Proof: We use a hybrid approach of BDD techniques, backtracking and divide-and-conquer. The first idea is to divide the board into two 4×8 boards. But then we obtain too many cases. There are 32 squares, where we may switch from one part to the other. Hence, we use another divide technique. (The ideas of this proof are illustrated in Fig. 1.) The board is divided into two 5×8 boards sharing two rows. By L, M and U we denote the three lower rows (row 1, 2 and 3), the two middle rows (row 4 and 5) and the three upper rows (row 6, 7 and 8) resp. The subboard consisting of the five lower rows is denoted as lower half board LM and the five upper rows as upper half board UM. A move from or to a square in L (or U) belongs to the lower (resp. upper) half board. Moreover, we define that a move from row 4 to row 5 belongs to the lower half board while a move from row 5 to row 4 belongs to the upper half board. Only the 16 squares of M can be reached by a move from L and left by a move to U or vice versa. We

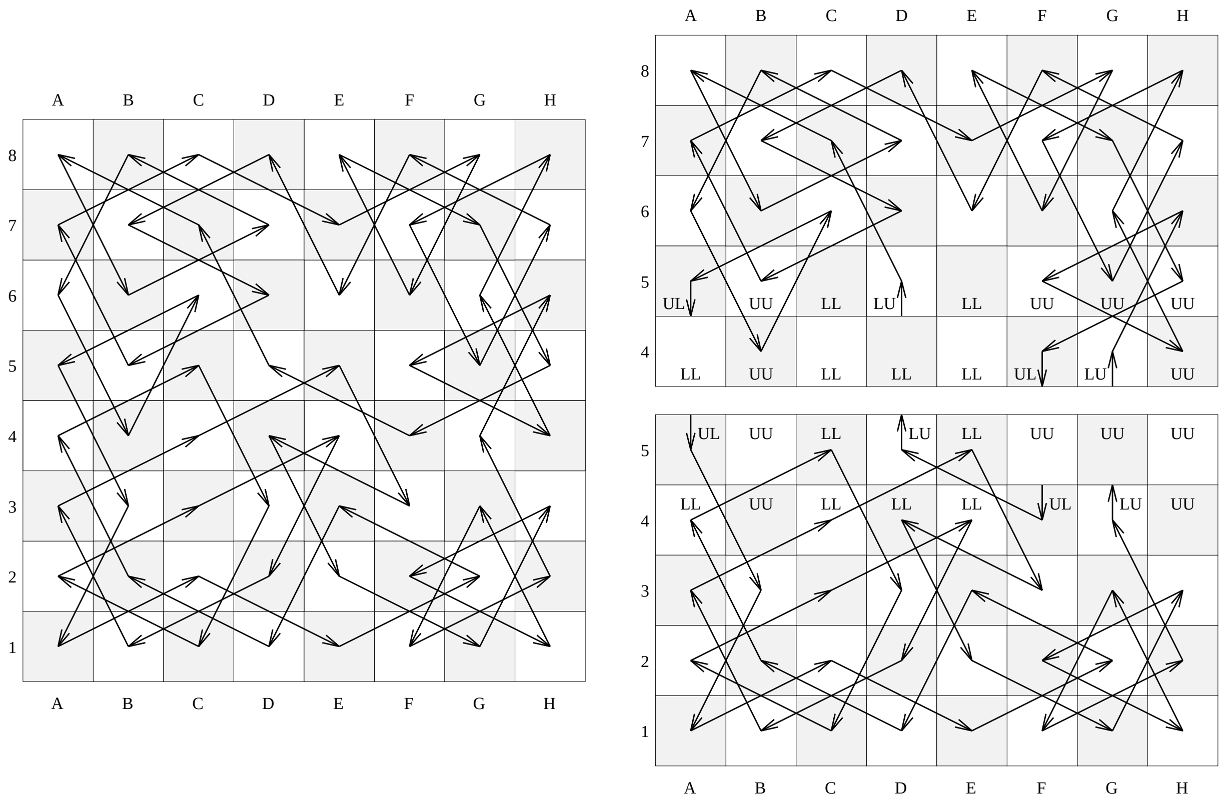

Figure 1: An example of a knight's tour

consider a knight's tour as a sequence of moves. Each tour is now divided into subtours (paths) belonging to the lower or the upper half board. We may divide our problem into the different cases with respect to this divide strategy. Each case will be described by properties assigned to the 16 squares of M.

We describe a case by a tuple (UL, LU, LL, UU) of sets of squares of M. UL denotes the squares reached by a move from the upper half board and left by a move to the lower half board, in LU the roles of L and U are exchanged. LL denotes the squares reached and left by moves of the lower half board, similarly for UU. In our example $UL = \{A5, F4\}$, $LU = \{D5, G4\}$, $LL = \{A4, C4, C5, D4, E4, E5\}$, and $UU = \{B4, B5, F5, G5, H4, H5\}$. In general, $|UL| = |LU|$ and $UU = M - (UL \cup LU \cup LL)$.

We have considered the following cases. For each $i \in \{1, \dots, 8\}$ we have chosen each pair (A, B) of subsets of M of size i. Then $A \cap B$ is interpreted as LL, $A - A \cap B$ as UL and $B - A \cap B$ as LU. Hence, we consider all cases, where $|LL| \leq 8$. Each case, where $|LL| \geq 9$, is symmetric to a case, where $|UU| \geq 9$. Hence, we multiply the number of solutions of those cases, where $|UU| \geq 9$, by 2. The number or considered cases equals

$$\sum_{1 \leq i \leq 8} \binom{16}{i}\binom{16}{i} = 383,358,644.$$

What are the subproblems in the lower and upper half board? Do we obtain knight's tours by combining solutions of the two subproblems?

In the lower half board a knight's tour consists of disjoint paths connecting squares from UL with squares from LU. Hence, each knight's tour belongs to a subset of tours with the same one-to-one mapping $f : UL \rightarrow LU$, indicating which squares are connected by the paths in the lower half board. The same arguments hold for the upper half board leading to one-to-one mappings $g : LU \rightarrow UL$.

Let $\#(UL, LU, LL, UU, f)$ be the number of different disjoint path systems respecting the partition (UL, LU, LL, UU) and connecting the squares of UL with the squares of LU in the way described by f. In particular, the squares of L, UL, LU and LL have to be covered exactly once without cycles. Let $\#(UL, LU, LL, UU, g)$ be the corresponding number for the upper half board.

There are cases, where f and g fit together, and cases, where we obtain no knight's tour. Let $UL = \{v_1, \ldots, v_k\}$ and $LU = \{w_1, \ldots, w_k\}$. The functions f and g describe pointers from UL to LU resp. from LU to UL. Since the outdegree of each square equals one, we obtain a unique path leaving v_1. If this path visits all other squares, we obtain $\#(UL, LU, LL, UU, f) \cdot \#(UL, LU, LL, UU, g)$ knight's tours by combining the path systems. This follows directly from the fact that we obtain a cycle on $UL \cup LU$ by following f and g. Otherwise we obtain cycle coverings with at least two cycles. Let us call (f, g) good, if they define one cycle on $UL \cup LU$. Knowing all numbers $\#(UL, LU, LL, UU, f)$ and $\#(UL, LU, LL, UU, g)$ the number of undirected knight's tours can be computed by

$$\frac{1}{2} \sum_{(UL,LU,LL,UU)} \sum_{(f,g) \text{ good}} \#(UL, LU, LL, UU, f) \cdot \#(UL, LU, LL, UU, g).$$

The factor $\frac{1}{2}$ is necessary, since otherwise we compute the number of directed knight's tours. It is easily possible to decide, whether (f, g) is good. In our example the knight's tour leads to $f(A5) = G4$, $f(F4) = D5$, $g(G4) = F4$ and $g(D5) = A5$ and, therefore, the cycle $A5 - G4 - F4 - D5 - A5$. The other one-to-one mapping $g'(G4) = A5$ and $g'(D5) = F4$ leads together with f to two cycles $A5 - G4 - A5$ and $F4 - D5 - F4$.

Our approach ensures that we do not have to list all knight's tours. It remains to describe how we obtain the numbers $\#(UL, LU, LL, UU, f)$ (and similarly the numbers $\#(UL, LU, LL, UU, g)$).

Here we work with MDD techniques to simplify "negative replacements". The lower half board is partitioned into the set of the 20 white squares and the set of the 20 black squares. The variables describe the possible moves. One MDD describes the function which outputs 1, if each white square is left once and each black square is reached exactly once. With simple replacement operations it then is ensured that the partition (UL, LU, LL, UU) is respected. The other MDD does the same for the moves from the black to the white squares. In particular, we are done, if one MDD outputs 0. Otherwise, we use backtracking techniques on the MDDs to count how many inputs lead to disjoint path segments without cycles. Each such input has to be counted for that parameter $\#(UL, LU, LL, UU, f)$ such that f describes the corresponding function between UL and LU. Again, the use of MDD/BDD techniques prevents the consideration of a lot of deadlocks.

The computation of the number of undirected knight's tours has been performed with 20 SUN workstations. Because of the partition of the problem in many disjoint subproblems the "parallelization" of the algorithm is obvious. The total run time is approximately four months. The CPU time is much less, since we could use the computers only during their idle times. $\square$

Again we ask whether we can trust in Theorem 2. We have proved that our approach is correct. Independent use of this approach can increase confidence in the number we have computed. Without performing this time consuming task the reader can verify the approach and the reader should be convinced that the number of knight's tours can be computed with this approach in reasonable time and space. Finally, the reader should be convinced, that BDD techniques are appropriate for the solution of concrete combinatorial problems.

6. ASYMPTOTIC RESULTS

Finally, we ask whether BDD techniques can help to obtain bounds on $T(n)$ for large n. Let $t(n) := T(n)^{1/n^2}$ for even n. Kyek, Parberry, and Wegener (1994) have proved with combinatorial methods and without BDD techniques that $t(n) \leq 4$ for all n and $t(n) \geq 1.3535$ for large n. Furthermore, we have that $t(2) = 0$, $t(4) = 0$, $t(6) = 1.2910\ldots$ and $t(8) = 1.6267\ldots$

With basic methods it can be proved that $t(n)$ converges to some limit t^*. But we are not able to prove that $t(n)$ is monotone increasing. Hence, the result on $t(8)$ does not lead directly to results on t^*.

With methods similar to those used in Section 5 we can solve the following problem on 8×8 chessboards with BDD techniques. For the three white squares W_1, W_2, W_3 in the top row without the corners we count the Hamiltonian paths from W_i to one of the 8 black squares in the two leftmost columns (case l), in the two rightmost columns (case r) and in the two bottom rows (case b). Let W_{ij}, $i \in \{1, 2, 3\}$, $j \in \{l, r, b\}$, be the corresponding number and W^* the minimum of all W_{ij}.

Lemma 1:

$$t^* \geq (W^*)^{1/64}.$$

Proof: Partition the $n \times n$ chessboard for large n into 8×8 subboards. At the border we take some larger rectangular boards. Then choose a king's tour without diagonal moves where the subboards are treated as squares. This king's tour determines in which order the subboards are visited. Let us assume that we reach a subboard at one of the 3 white squares of the first row. Then there exist W^* Hamiltonian paths such that the next board on the king's tour can be reached at some square at the border but not in a corner. This gives $(W^*)^{n^2/64}$ Hamiltonian tours, if n is divisible by 8. $\square$

We have not computed W^* with the computer but we conjecture that the result is close to $8T(8)$, since we have the choice between 8 terminal nodes. This would imply $t^* \geq 1.6804$. Hence, BDD techniques may lead even to improved asymptotic results. Finally, we remark that we can improve the lower bounds on t^* by considering different king's tours without diagonal moves. BDD techniques are adequate also for these purposes.

Conclusion

Binary decision diagrams which have been developed for hardware verification and computer-aided design can also be applied to the solution of combinatorial problems. This has been proved by the solution of some open combinatorial chess problems. The reason for such a success is that OBDDs and ZBDDs recognize automatically isomorphic cases and situations without any solution.

REFERENCES

K. Appel and W. Haken (1977). Every planar map is four colorable, part I: discharging. *Illinois Journal of Mathematics* (21), 429–490.

K. Appel, W. Haken, and J. Koch (1977). Every planar map is four colorable, part II: reducibility. *Illinois Journal of Mathematics* (21), 491–567.

B. Bollig, P. Savicky, and I. Wegener (1994). On the improvement of variable orderings for OBDDs. *IFIP Workshop on Logic and Architecture Synthesis*, 71–80.

K. S. Brace, R. L. Rudell, and R. E. Bryant (1990). Efficient implementation of a BDD package. *Proc. 27. Design Automation Conference*, 40–45.

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

R. E. Bryant (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys 24*(3), 293–318.

A. Conrad, T. Hindrichs, H. Morsy, and I. Wegener (1994). Solution of the knight's Hamiltonian path problem on chessboards. *Discrete Applied Mathematics 50*, 125–134.

R. L. Graham and V. Rödl (1987). In Whitehead, C. (ed.): Numbers in Ramsey theory. *Surveys in Combinatorics, London Math. Soc. Lecture Notes 123*, 111–153.

J. E. Hopcroft and J. D. Ullman (1979). *Introduction to automata theory, languages, and computation.* Addison-Wesley, New York.

O. Kyek, I. Parberry, and I. Wegener (1994). Bounds on the number of knight's tours. Technical Report 555, Universität Dortmund.

Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula (1994). EVBDD-based algorithms for integer linear programming, spectral transformation and function decomposition. *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems 13*(8), 959–975.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmyos (1985). *The traveling salesman problem. A guided tour of combinatorial optimization.* Wiley, New York.

B. D. McKay and S. P. Radziszowski (1991). The first classical Ramsey number for hypergraphs is computed. *ACM-SIAM SODA Proc.*, 304–308.

B. D. McKay and S. P. Radziszowski (1994). Linear programming in some Ramsey problems. *Journal of Combinatorial Theory (B) 61*, 125–132.

S.-I. Minato (1993). Zero-suppressed BDDs for set manipulation in combinatorial problems. *Proc. of the 30th ACM/IEEE Design Automation Conference*, 272–277.

S.-I. Minato (1994). Calculation of unate cube set algebra using zero-suppressed BDDs. *Proc. of the 31st ACM/IEEE Design Automation Conference*, 420–424.

S. Panda and F. Somenzi (1995). Who are the variables in your neighborhood. *International Workshop on Logic Synthesis*, 5/11–5/20.

W. W. Rouse Ball and H. S. M. Coxeter (1987). *Mathematical recreations and essays.* Dover, New York.

I. Semba and S. Yajima (1994). Combinatorial algorithms using Boolean processing. *Trans. of the Information Processing Society of Japan 35*(9), 1661–1673.

H. C. Warnsdorff (1823). *Des Rösselsprunges einfachste und allgemeinste Lösung.* Schmalkalden.

I. Wegener (1994). Efficient data structures for Boolean functions. *Discrete Applied Mathematics 136*, 347–372.