IMVI ISSN (p) 2303-4882, ISSN 1840-4383
Open Mathematical Education Notes
Vol. 5 (2015), special issue, 1-30
www.imvibl.org / JOURNALS / IMVI OMEN

THE SOLUTION OF COMBINATORIAL PROBLEMS USING BOOLEAN EQUATIONS: NEW CHALLENGES FOR TEACHING

Bernd Steinbach
Freiberg University of Mining and Technology, Germany
steinb@informatik.tu-freiberg.de

Christian Posthoff
The University of the West Indies, Trinidad & Tobago
christian@posthoff.de

Abstract

It will be shown that many finite combinatorial problems can be solved by using *Boolean equations*. Therefore, it is necessary to teach how to transform the different problems into this area in a *correct* way. We demonstrate the required modeling steps in the first part using very different interesting examples. It is no longer necessary to develop different solution methods for different tasks; Boolean equations can uniquely be used. *Ternary vectors* and *ternary arrays* are powerful data structures that help to extenuate the complexity.

Naturally, problems of a reasonable size cannot be solved by hand, therefore existing software systems must be used. The application of such a software system can require a huge amount of details. We show how a simplified environment of the software supports the teaching process because the key issues can be explored on a reasonable level of abstraction. Both the *XBOOLE-Monitor* and a *SAT-solver* will be used. It must be learned how to use these systems, i.e., the structure and the working of the systems must be well understood.

It can, however, be that the complexity of the problem is beyond the size of the existing systems. Then we have the last level that requires an excellent knowledge of Mathematics and excellent programming skills. We take as an example the rectangle-free coloring of grids using four colors. The solution of these last problems of the highest level show that only the unified efforts of Mathematics and Computer Science can solve the problems of the highest complexity. It is possible to solve problems that could not be solved before in a constructive way. You do not only know whether solutions exist or not, it is possible to build a very large number of solutions or even all the solutions.

Therefore, the education of Mathematicians, Computer Scientists and Engineers must take this necessary cooperation into consideration. To be a specialist in one of these areas is not enough!

Key words: Artificial Intelligence; Chessboard Problems; Graph Coloring; Four-valued Edge Coloring; Complete Bipartite Graph; Rectangle-free Grid; Boolean Equation; SAT-solver; Sudoku; Ternary Vector; XBOOLE; XBOOLE-Monitor; Challenges for Teaching
ZDM Subject Classification: D50, N70, N80, R20, R40

1. Introduction

We want to give here a summary of our longstanding research in the area of Boolean (binary, logic) equations and reflect the solution methods on teaching requirements. There are, based on the results of this research, several points which we want to bring up for discussion.

- It will be shown that many combinatorial problems [10, 13, 15, 18, 19] can be transformed into satisfiability problems (shortly SAT) and solved using the developed models and algorithms. This approach is constructive and very general, no search procedures are involved.
- In many cases, it is not necessary to write down the huge number of clauses of the conjunctive forms which must be solved by a SAT-solver [2]. Based on the explored properties of the problem, it is possible to generate partial solution sets of the restrictive properties of the problem.
- As extension of the classical SAT-approach we suggest a new modeling procedure: a *two-phase SAT-solver* [12] which utilizes the efficient operations of the programming system XBOOLE [14, 18]. In the first phase this SAT-solver creates partial solution sets which are used in the second phase to calculate the final solution without any further testing or special decisions.
- In many applications the Boolean modeling can be considered as very efficient, and it is not necessary to develop special algorithms. It is much easier to use a general methodology based on Boolean equations and ternary vectors.
- Search algorithms disappear more or less completely. The final solutions will be built in a constructive way.
- In recent years the power of SAT-solvers has been improved. The problems to solve must be mapped into a single, but often very large, Boolean equation where a conjunctive form is equal to 1.

The main part of this paper is organized as follows. Section 2. introduces the used Boolean operations and ternary vectors as the main data structure. Section 3. explains the aim of SAT-solvers and shows how this task can be solved using ternary vectors. As a first example, we explain in Section 4. how the Sudoku game can be modeled by Boolean equations and solved using XBOOLE and the method of a two-phase SAT-solver. Section 5. introduces problems on a chessboard and encouraged the reader to solve these problems in a similar manner. A wide field of application of the Boolean problems belongs to graphs. Section 6. shows how both the requirements and the constraints of a graph-coloring problem can be modeled and solved using the same approach.

Table 1. Boolean operations: (a) unary, (b) binary

(a)

	complement (NOT)
x	$\overline{x}$
0	1
1	0

(b)

		conjunction (AND)	disjunction (OR)	antivalence (EXOR)	implication (IF ... THEN)
x	y	$x \wedge y$	$x \vee y$	$x \oplus y$	$x \rightarrow y$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	1

The exponential complexity in the Boolean domain is a strong challenge in both research and teaching. Section 7. introduces the problem of the rectangle-free coloring of grids using 4 colors and faces us with the gigantic number of more than 10^{195} different color patterns in which a pattern with a special property is searched. Section 8. shows that despite of this extreme complexity a binary model can be established. Section 9. explains some basic approaches to solve this coloring problem up to a certain size of the complexity and shows how the knowledge of the detected restrictions can be utilized for further improvements. Four steps to the final solution of this very complex problem are reported in Section 10..

Before we conclude this paper in Section 12., we summarize in Section 11. our teaching experiences in the Boolean domain and point out some challenges for the teaching process in this domain caused by the strong progress in real applications.

2. Boolean Operations and Ternary Vectors as the Main Data Structure

This paper deals with Boolean values of the set $\mathbb{B} = \{0,1\}$ and uses the usual Boolean operations of Table 1. The $\wedge$-sign of the conjunction will often be left out like the multiplication sign in the field of mathematics.

As the next step we introduce the data structure of a *ternary vector*. Let

$$\mathbf{x} = (x_1, \cdots, x_n),\ x_i \in \{0,1,-\},\ i = 1,\ldots,n\ ,$$

then $\mathbf{x}$ is called a *ternary vector* (TV) which can be understood as an abbreviation of a set of binary vectors. When we replace each $-$ by both 0 and 1, then we get several binary vectors *generated* by this ternary vector. In this way, the vector $(0-1-)$ represents four binary vectors (0010), (0011), (0110), and (0111). A list (matrix) of ternary vectors (TVL) can be understood as the union of the corresponding sets of binary vectors.

There is a direct relation of ternary vectors with conjunctions of Boolean variables. When a conjunction C with variables $x_1,\ldots,x_k$ is given, then we can build a ternary vector t with the components $t_1,\ldots,t_k$ according to the following coding:

$$\begin{aligned} x_i: &\quad t_i = 1\ , \\ \overline{x}_i: &\quad t_i = 0\ , \\ x_i \text{ missing}: &\quad t_i = -\ . \end{aligned} \tag{1}$$

This coding expresses directly on one side the respective conjunction, on the other side the set of all binary vectors satisfying $C = 1$.

Example 1 *Let be given* $x_1\overline{x}_2x_3\overline{x}_5 = 1$*, then we have* $\boldsymbol{t} = (101-0)$ *which expresses the two binary vectors* (10100) *and* (10110).

Hint: It will be assumed that the problem-relevant Boolean space includes the variables x_1, x_2, x_3, x_4, x_5.

Let be given two ternary vectors **x** and **y**. The intersection of these two vectors (i.e., the intersection $(\cap)$ of the respective two sets of binary vectors) will be computed according to Table 2 which has to be applied in each component of the two vectors. The symbol $\emptyset$ indicates that the intersection of the two sets is empty. A sophisticated coding of the three values 0, 1 and $-$ allows

Table 2. Intersection of ternary values

x_i	0	0	0	1	1	1	$-$	$-$	$-$
y_i	0	1	$-$	0	1	$-$	0	1	$-$
$x_i \cap y_i$	0	$\emptyset$	0	$\emptyset$	1	1	0	1	$-$

the introduction of binary vector operations that can be executed on the level of registers (32, 64 or even 128 bits in parallel). We use the coding of Table 3. The first bit indicates that the variable has a value 0 or 1 in the ternary vector, the second bit indicates the value itself.

Table 3. Binary code of ternary values

ternary value	bit1	bit2
0	1	0
1	1	1
$-$	0	0

When the three-valued operations for the intersection are transferred to these binary vectors, then the intersection is empty iff

$$bit1(\mathbf{x}) \wedge bit1(\mathbf{y}) \wedge (bit2(\mathbf{x}) \oplus bit2(\mathbf{y})) \neq \mathbf{0} \,. \tag{2}$$

If the intersection is not empty, then it can be determined by the following bit vector operations:

$$bit1(\mathbf{x} \cap \mathbf{y}) = bit1(\mathbf{x}) \vee bit1(\mathbf{y}) \,, \tag{3}$$

$$bit2(\mathbf{x} \cap \mathbf{y}) = bit2(\mathbf{x}) \vee bit2(\mathbf{y}) \,. \tag{4}$$

Hint: $\mathbf{0}$ is the vector where all the components are equal to 0. Hence, by using some very fast and very simple bit vector operations (available on the hardware level), we can find the intersection of two ternary vectors.

3. Basic Approaches of Parallel SAT-Solving

It is the aim of a SAT problem to find at least one assignment of Boolean variables such that a Boolean expression in conjunctive form becomes true. A conjunctive form means that disjunctions (also called clauses) of Boolean variables (some so them can be negated) are connected by AND-operations. Using ternary vectors as the basic data structure, we are able to calculate all solutions of SAT-problems directly. We will use the following small example:

$$(a \vee \overline{b} \vee \overline{c})(b \vee \overline{d} \vee \overline{e})(\overline{a} \vee d \vee e)(b \vee c \vee \overline{e}) = 1 \; . \tag{5}$$

This equation is equivalent to the system of four single equations:

$$a \vee \overline{b} \vee \overline{c} = 1 \; , \tag{6}$$
$$b \vee \overline{d} \vee \overline{e} = 1 \; , \tag{7}$$
$$\overline{a} \vee d \vee e = 1 \; , \tag{8}$$
$$b \vee c \vee \overline{e} = 1 \; . \tag{9}$$

The first equation (6) now will be transformed into the following ternary matrix (a *set*, or *list*, or *array* of ternary vectors):

a	b	c	d	e
1	–	–	–	–
0	0	–	–	–
0	1	0	–	–

.

This matrix shows all the vectors that satisfy Equation (6). If $a = 1$, then the values of the other variables are not important. Alternatively, Equation (6) is satisfied if $a = 0$ and $b = 0$. Finally, if $a = 0$ and $b = 1$, then c must be equal to 0 in order to satisfy Equation (6). This construction has the additional property (advantage) that every pair of vectors of this matrix has an empty intersection, they are *orthogonal to each other*, and therefore any double solutions cannot exist. $\overline{b}$ indicates the negation of b. It is very characteristic that each vector of the matrix includes more information than the previous vectors. The number of vectors in the resulting matrix is equal to the number of variables in the disjunction. In the example, each disjunction has three variables. If we repeat this procedure for all four equations, then we get the following four matrices as partial solution sets S_i:

$S_1 =$

a	b	c	d	e
1	–	–	–	–
0	0	–	–	–
0	1	0	–	–

solution of Equation (6),

$S_2 =$

a	b	c	d	e
–	1	–	–	–
–	0	–	0	–
–	0	–	1	0

solution of Equation (7),

$S_3 =$

a	b	c	d	e
0	–	–	–	–
1	–	–	1	–
1	–	–	0	1

solution of Equation (8), and

$$S_4 = \begin{array}{|ccccc|}\hline a & b & c & d & e \\ \hline - & 1 & - & - & - \\ - & 0 & 1 & - & - \\ - & 0 & 0 & - & 0 \\ \hline\end{array} \quad \text{solution of Equation (9).}$$

In order to get the final solution, these four matrices have to be combined by intersection (see above). Each line of one matrix has to be combined with each line of the next matrix, empty intersections can be omitted.

For the first and second matrix, we get, for instance, after some simplifications:

$$S_1 \cap S_2 = \begin{array}{|ccccc|}\hline a & b & c & d & e \\ \hline 1 & - & - & - & - \\ 0 & 0 & - & - & - \\ 0 & 1 & 0 & - & - \\ \hline\end{array} \bigcap \begin{array}{|ccccc|}\hline a & b & c & d & e \\ \hline - & 1 & - & - & - \\ - & 0 & - & 0 & - \\ - & 0 & - & 1 & 0 \\ \hline\end{array} = \begin{array}{|ccccc|}\hline a & b & c & d & e \\ \hline 1 & 1 & - & - & - \\ 0 & 1 & 0 & - & - \\ - & 0 & - & 0 & - \\ - & 0 & - & 1 & 0 \\ \hline\end{array},$$

and the final solution is equal to

$$S = \bigcap_{i=1}^{4} S_i = \begin{array}{|ccccc|}\hline a & b & c & d & e \\ \hline 0 & 0 & - & - & 0 \\ - & 0 & 1 & 0 & 1 \\ 1 & 1 & - & - & 1 \\ 0 & 1 & 0 & - & - \\ 1 & - & - & 1 & 0 \\ \hline\end{array}. \tag{10}$$

This matrix of ternary vectors represents all solutions of the original SAT-problem. Since the value $-$ represents 0 as well as 1, the equation has 18 solutions represented by five ternary vectors.

4. Sudoku as an Example

In recent years a Japanese game with the name *Sudoku* became very popular. It is played mostly on a board with 9×9 cells, but other square numbers are also possible, such as 4×4 or 16×16 or even 25×25. It is easy to understand and a bit challenging for human beings, and it can be used comfortably to spend waiting time on airports or similarly. But there are also mathematical and logical properties that deserve some attention.

There is a square board of, for instance, the size 9×9. The values $1, \ldots, 9$ have to be set such that in each column, in each row and in each subsquare of size 3×3 each value is used once and only once.

Some values already have been set; these values are the clues. The other values have to be found according to the existing values. We enumerate the columns from the left to the right and the rows top-down (as usual for a matrix).

We know at least two papers [8, 25] that are using SAT for the modeling of the game and existing SAT-solvers for the solution of the problem. These papers used the following approach: a binary variable x_{ijk} describes the content of the cell (i, j), where i is the index of the row and j is the index of the column, and $1 \leq i, j, k \leq 9$:

$$x_{ijk} = \begin{cases} 1 & \text{if the value in the cell } (i, j) = k \\ 0 & \text{if the value in the cell } (i, j) \neq k \end{cases} \tag{11}$$

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 1. Example of a 9×9 Sudoku.

The transformation into a SAT-problem uses several steps:

$$x_{ij1} \vee x_{ij2} \vee x_{ij3} \vee x_{ij4} \vee x_{ij5} \vee x_{ij6} \vee x_{ij7} \vee x_{ij8} \vee x_{ij9} = 1 \ . \tag{12}$$

expresses the requirement that one of the numbers $1, \ldots, 9$ must be used for the cell (i, j). Such a disjunction must be written for each cell of the board which results in 81 clauses which must be satisfied simultaneously.

The second step expresses all the constraints for rows, columns and squares as clauses as well. For example, for the cell $(1,1)$ and the value 1 in this cell, no other value can be in this cell:

$$x_{111} \rightarrow \overline{x}_{112}, x_{111} \rightarrow \overline{x}_{113}, \ldots, x_{111} \rightarrow \overline{x}_{118}, x_{111} \rightarrow \overline{x}_{119} \ . \tag{13}$$

The same set of clauses must be written for the other values $2, \ldots, 9$ in the same cell. The constraints for the first row can be expressed in the same way:

$$x_{111} \rightarrow \overline{x}_{121}, x_{111} \rightarrow \overline{x}_{131}, \ldots, x_{111} \rightarrow \overline{x}_{181}, x_{111} \rightarrow \overline{x}_{191} \ . \tag{14}$$

The constraints for the first column are given as:

$$x_{111} \rightarrow \overline{x}_{211}, x_{111} \rightarrow \overline{x}_{311}, \ldots, x_{111} \rightarrow \overline{x}_{811}, x_{111} \rightarrow \overline{x}_{911} \ , \tag{15}$$

and finally, we must consider the remaining value 1 in the respective square:

$$x_{111} \rightarrow \overline{x}_{221}, x_{111} \rightarrow \overline{x}_{231}, x_{111} \rightarrow \overline{x}_{321}, x_{111} \rightarrow \overline{x}_{331} \ . \tag{16}$$

Again all these clauses have to be written for all numbers from 1 to 9 and finally for all cells. By using the rule $x \rightarrow y = \overline{x} \vee y$ the whole set of implications can be transformed into disjunctions (clauses). The number of all restrictive clauses is equal to $9 * 81 * (8 + 8 + 8 + 4) = 20,412$. All of them must be satisfied together with the single variable clauses of the clue and the 81 clauses of the requirements at the same time, and this is the problem in SAT-format. Each satisfying set of values for the binary variables is a solution of the Sudoku.

We will show that this game easily can be modeled by using a logic equation, with *ternary vectors* as the most appropriate data structure. Actually, the logic equation does not even have to be written down, *the ternary vectors can be generated directly*.

The constraints can be stated by one single conjunction for each number on each cell:

$$C_{111} = x_{111} \wedge \overline{x}_{112}\overline{x}_{113}\overline{x}_{114}\overline{x}_{115}\overline{x}_{116}\overline{x}_{117}\overline{x}_{118}\overline{x}_{119} \wedge \overline{x}_{121}\overline{x}_{131}\overline{x}_{141}\overline{x}_{151}\overline{x}_{161}\overline{x}_{171}\overline{x}_{181}\overline{x}_{191} \wedge \overline{x}_{211}\overline{x}_{311}\overline{x}_{411}\overline{x}_{511}\overline{x}_{611}\overline{x}_{711}\overline{x}_{811}\overline{x}_{911} \wedge \overline{x}_{221}\overline{x}_{231}\overline{x}_{321}\overline{x}_{331} \; . \tag{17}$$

This conjunction describes completely the setting of the value 1 on the cell $(1,1)$ and *all the consequences*. There are 729 of such conjunctions which are defined uniquely. It is important to understand that not only the requirements in terms of 9 variables x_{ijk} are taken into consideration, but the conjunctions C_{ijk} so that *all the consequences* resulting from a given setting are used immediately. The existing knowledge or constraints are directly built into the ternary vectors.

Now we must express the possibilities of the game. In order to do this, we can use one of the following four types of equations.

1. The equation

$$C_{111} \vee C_{112} \vee C_{113} \vee C_{114} \vee C_{115} \vee C_{116} \vee C_{117} \vee C_{118} \vee C_{119} = 1 \tag{18}$$

   describes that one of the nine values must be assigned to one cell (the cell (1,1) is only an example).

2. The equation

$$C_{111} \vee C_{121} \vee C_{131} \vee C_{141} \vee C_{151} \vee C_{161} \vee C_{171} \vee C_{181} \vee C_{191} = 1 \tag{19}$$

   describes that the value 1 must be assigned to one of the cells in a row (row 1 and value 1 are only examples).

3. The equation

$$C_{111} \vee C_{211} \vee C_{311} \vee C_{411} \vee C_{511} \vee C_{161} \vee C_{711} \vee C_{811} \vee C_{911} = 1 \tag{20}$$

   describes that the value 1 must be assigned to one of the cells in a column (column 1 and value 1 are only examples).

4. The equation

$$C_{111} \vee C_{121} \vee C_{131} \vee C_{211} \vee C_{221} \vee C_{231} \vee C_{311} \vee C_{321} \vee C_{331} = 1 \tag{21}$$

   describes that the values 1 must be assigned to one of the cells in a subsquare (the first subsquare and value 1 are only examples).

Each type of these equations generates a system of 81 disjunctions that must be satisfied at the same time. They are completely equivalent [19], one system can be selected once and for ever. Each disjunction consist of nine conjunctions of 29 variables. All the conjunctions are represented by ternary vectors, and *this representation can be generated before any real game* which is given by special settings. Each ternary vector will have 729 components, and all intersections from the left to the right have to be calculated. Based on (18), we get as a general model of a 9×9 Sudoku the Boolean equation:

$$\bigwedge_{i=1}^{9} \bigwedge_{j=1}^{9} \left(\bigvee_{k=1}^{9} C_{ijk} \right) = 1 \; . \tag{22}$$

Three similar equations follow from (20), (19), and (21).

The solution of (22) consists of 6,670,903,752,021,072,936,960 binary vectors of all valid 9×9 Sudokus [4]. It is not a good idea to calculate such a huge amount of solution vectors and restrict this set by the clue. A better way is to start calculation of the solution with the knowledge of the clue.

$$\bigwedge_{ijk \in \text{clue}} C_{ijk} = 1 \ . \tag{23}$$

The AND-operation of (23) can be calculated using the intersection-operation `ISC` of XBOOLE [10, 14, 16]. The result of (23) is a *single ternary vecto*r in which the variables x_{ijk} of the clue are equal to 1, the caused restrictions are manifested by $x_{ijk} = 0$, and the remaining unknown variables carry dash $(-)$ values. This start-up procedure significantly reduces both the search space and the solution time.

The solution of a special Sudoku is defined by the system of equations (23) and (22). After solving Equation (23), 81 further intersection of Equation (22) must be calculated where each intersection uses a list of 9 ternary vectors which are specified by the expression in parentheses of (22). Due to the commutativity of the conjunction, the result does not depend on the order of these 81 intersections. However, the time for the solution can be significantly reduced when for the calculation of (22) the cells (i, j) with a small number of remaining dashes are preferred [19].

The solution set S consists of all binary vectors of the length 729 that solve the SAT-problem of the given Sudoku. Each solution vector includes exactly 81 values 1 which indicate the solution numbers associated to the cells; the remaining $729 - 81 = 648$ components of the solution vector are equal to 0. Thus, by taking the index (i, j, k) of the values 1 in the solution vector, a representation of the value k in the cell (i, j) of row i and column j can be established.

As a summary, we can see that the solution of the problem has two steps.

1. The first phase covers the modeling of the problem and the calculation of partial solution sets (or solution candidates). Of course, the first phase depends on the problem to be solved - in our case any Sudoku game.

2. The second phase mainly considers the different action possibilities and combines these possibilities by $\vee$ to partial solution sets. The intersections of these partial solution sets lead to the final solution.

The advantages of this new approach in comparison with the known traditional SAT-models can be summarized as follows.

- Solution values are assigned in each step to many variables instead of a single variable by a SAT-solver. In the case of a 9×9 Sudoku a single assignment specifies additionally 28 variables of the solution space, and this strongly restricts the remaining search space.

- The knowledge about the problem can be expressed in a compact manner without the need of distributing to many clauses. The basic requirement of the SAT-solver to prepare the task in a conjunctive form can be skipped. In the case of a 9×9 Sudoku the general requirements can be expressed by the conjunction of 81 disjunctive forms of 9 conjunctions each instead of $81 + 20,412 = 20,493$ clauses in a conjunctive form for the SAT-solver.

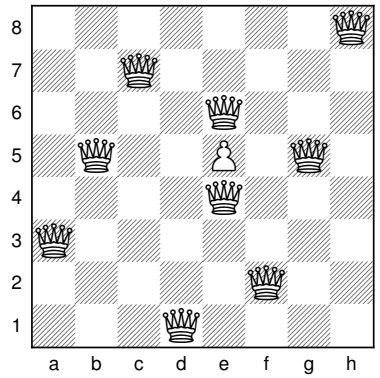

Figure 2. Example of a solution of 9 queens and 1 pawn.

In the case of a 16×16 board the matrix of the partial solution sets required approximately 2 Megabyte. Each row of this matrix includes one value 1, 54 values 0. The remaining values of the 4,096 variables are filled with dashes. Therefore, we decided to store only the index values of the elements with the value 0 and 1 and to generate any vector of a partial solution set at the time when it is required. Without any other changes the problem of a 16×16 Sudoku that maps into a problem of 4,096 variables and 111,616 clauses could be solved within about two and a half minutes.

Alternative approaches to solve a difficult Sudoku were recently published in [19]. Using an XBOOLE-implementation which utilized the graphics processing unit (GPU) [23] the 44,664 solutions of a 25-clue Sudoku were found in less than one second which is more than 20 times faster than the SAT-solver clasp-2.0.0-st-win32 [6].

5. Chessboard Problems

Based on this methodology, many other problems have been solved. It will not be very difficult to apply the same methodology.

1. It is expected that on a chessboard of size $n \times n$ with k additional pawns $n+k$ queens can be placed without threatening each other. Figure 2 shows one solution for one pawn on a board of the size 8×8.

   The pawn interrupts the effective lines of the queens, and the diagram really shows nine queens on this board. Figure 3 shows the number of solutions depending on the position of the pawn.

   Figure 4 shows finally the result for two pawns, and Table 4 summarizes the experimental results for chessboards of several sizes.

2. The case of $k = 0$ is the "normal" problem of arrangements of queens on a chessboard $n \times n$ that has been solved as well up to $n = 17$.

3. There are many problems asking for minimum and maximum numbers, for instance, how many bishops are *at least* required to cover all the fields on a chessboard, or how many

0	0	0	0	0	0	0	0
0	0	2	4	4	2	0	0
0	2	6	2	2	6	2	0
0	4	2	10	10	2	4	0
0	4	2	10	10	2	4	0
0	2	6	2	2	6	2	0
0	0	2	4	4	2	0	0
0	0	0	0	0	0	0	0

Figure 3. Distribution of the solutions of 9 queens and 1 pawn.

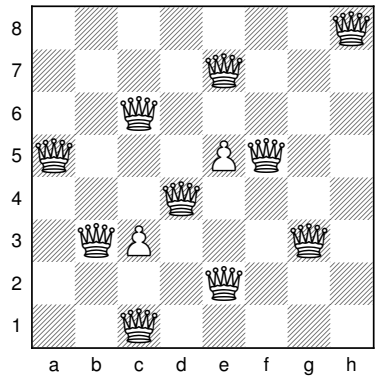

Figure 4. Example of 10 queens and 2 pawns.

Table 4. Number of variables and solutions for 2 pawns and the maximal number of queens on chessboards of the size $n \times n$

n	# variables	# solutions	time in seconds
3	9	0	0.00
4	16	0	0.00
5	25	0	0.00
6	36	0	0.00
7	49	4	0.20
8	64	44	0.92
9	81	280	3.31
10	100	1,304	11.09
11	121	12,452	97.21
12	144	105,012	406.07

bishops can at most be placed on a chessboard without threatening each other etc. These problems also have been solved on boards of size $m \times n$ for many values of m, n.

The readers are encouraged to find the Boolean equations which model these tasks. Having these equations they will recognize that the support of a computer is necessary for the solution of the systems of Boolean equations. We suggest to use the XBOOLE-Monitor for this final solution step. The XBOOLE-Monitor can be downloaded (for free) at

`http://www.informatik.tu-freiberg.de/xboole/` .

All information needed to use the XBOOLE-Monitor is given in the help system that is included in this software. Such tasks are easily understandable and are suitable for teaching the suggested method in classrooms.

6. Graph Coloring

It is, at a first glance, very surprising that also many graph problems can be solved in this way. The coloring of graphs is an area with a huge amount of publications. Our methods can be applied to color any graph, the nodes as well as the edges. As an example, we use the graph called *Birkhoff's Diamond* [11]. The coloring of this graph has been considered as very difficult.

The structure of a graph can be described using an adjacency matrix. A value 1 in the row i and column j indicates an edge from node i to node j in the graph. In the case of an undirected graph, we get a symmetric adjacency matrix. The graph *Birkhoff's Diamond* has the following adjacency matrix:

$$A_{BD} = \begin{pmatrix} 0100011100 \\ 1010000110 \\ 0101000010 \\ 0010100011 \\ 0001011001 \\ 1000101000 \\ 1000110101 \\ 1100001011 \\ 0111000101 \\ 0001101110 \end{pmatrix} . \tag{24}$$

Using this adjacency matrix (24) the partial solution sets can be generated directly. The logic variables describe whether a certain color is assigned to a node of the graph or not. Hence, the number of required variables is equal to the product of the number of nodes and considered colors.

$$x_{cn} = \begin{cases} 1 & \text{if the color } c \text{ is assigned to the node } n \\ 0 & \text{if the color } c \text{ is not assigned to the node } n \end{cases} . \tag{25}$$

As next we express restrictive rules for the color c and the node i by the conjunction C_{ci}. We take as an example the assignment of the color $c = 1$ to the node $i = 1$: in this case it is not allowed that:

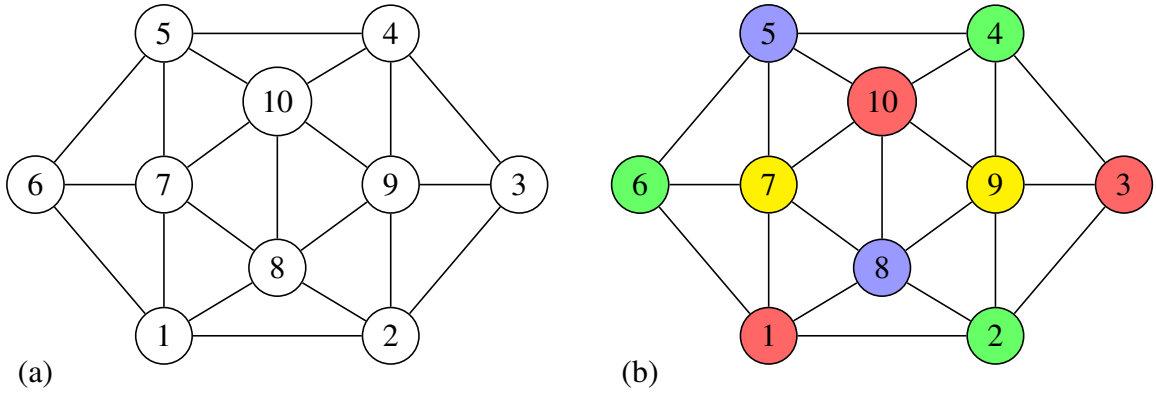

Figure 5. Birkhoff's diamond: (a) uncolored graph, (b) one solution using 4 colors.

1. another color $(c = 2,3,4)$ is assigned to the same node $(i = 1)$, and
2. the same color $(c = 1)$ is assigned to another node (taken from the first row in A_{BD}: $i = 2,6,7,8$) connected by an edge to the node $(i = 1)$.

In this way we get:

$$C_{11} = x_{11} \wedge \overline{x}_{21}\overline{x}_{31}\overline{x}_{41} \wedge \overline{x}_{12}\overline{x}_{16}\overline{x}_{17}\overline{x}_{18} \; . \tag{26}$$

The second phase of the new two-phase SAT-solver is controlled by the requirement clauses. For graph coloring we have the simple requirement that there must be one color assigned to each node of the graph. In order to find all allowed assignments of the four colors for the graph of Figure 5 (a), we must solve the equation:

$$\bigwedge_{i=1}^{10} (C_{1i} \vee C_{2i} \vee C_{3i} \vee C_{4i}) = 1 \; . \tag{27}$$

The time to solve this equation using the operations `UNI` and `ISC` of XBOOLE [10] was less than a single time-tick (15 ms). Figure 5 (b) shows one of the 576 solutions that have been found.

Two experiments demonstrate the power of this approach. In the first experiment we calculated all solutions to color Birkhoff's diamond using three, four or five colors. Table 5 summarizes these results.

Table 5. Calculation of all solutions to color Birkhoff's diamond using 3, 4 of 5 colors

number of				time in
nodes	colors	variables	solutions	seconds
10	3	30	0	0.00
10	4	40	576	0.00
10	5	50	40800	0.02

In the second experiment we created several larger graphs: we combined first two Birkhoff's diamonds using some additional edges and thereafter four Birkhoff's diamonds in a similar way [11]. Table 6 summarizes these results.

Table 6. Calculation of all solutions to color the Birkhoff's diamond and graphs that include two or four such graphs using 4 colors

number of				time in
nodes	colors	variables	solutions	seconds
10	4	40	576	0.00
20	4	80	99888	0.20
40	4	160	100800	4.97

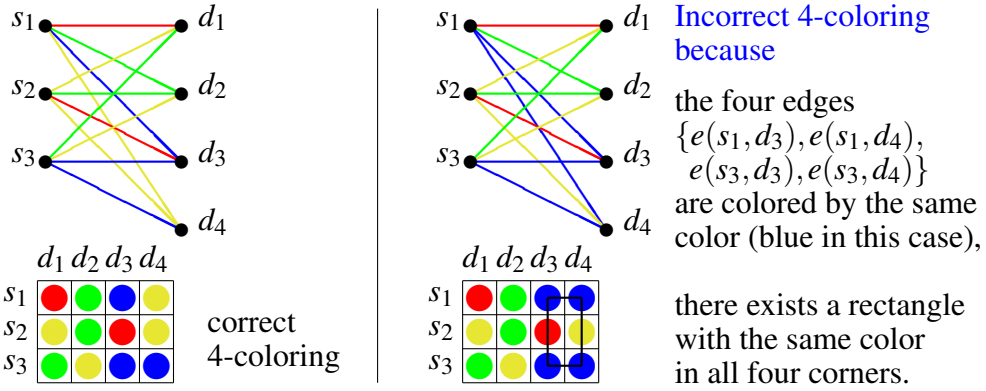

Figure 6. Edge coloring of complete bipartite graphs with four colors: (a) correct, (b) incorrect.

7. Grid Coloring

After we solved several graph problems based on the presented methodology rather easily, we found by chance the following grid coloring problem [5] (we exchanged m and n to get in natural order m rows and n columns):

> "A two-dimensional *grid* is a set $G_{m,n} = [m] \times [n]$. A grid $G_{m,n}$ is *c-colorable* if there is a function $\chi_{m,n} : G_{m,n} \to [c]$ such that there are no rectangles with all four corners the same color."

There are many practical tasks which can be modeled and solved by graph coloring [9].

Table 7. Knowlege about rectangle-free 4-colorable grids: C - 4-colorable, N - not 4-colorable, U - it is unknown whether this grid of this size is rectangle free 4-colorable

rows	columns											
	10	11	12	13	14	15	16	17	18	19	20	21
16	C	C	C	C	C	C	C	C	C	C	C	N
17	C	C	C	C	C	C	C	U	U	N	N	N
18	C	C	C	C	C	C	C	U	U	N	N	N
19	C	C	C	C	C	C	C	N	N	N	N	N

Such tasks are, for instance, the frequency assignment of radio stations, the aircraft scheduling to flights or the state assignment for optimizing finite-state machines. Graph coloring can be done by assigning colors either to the vertices (as shown above) or to the edges of a given graph. We focus in the following sections of this paper to the edge coloring of complete bipartite graphs. Such a graph consists of two disjoint sets of vertices: the source vertices s_i and the destination vertices d_j.

There are different data structures that represent a bipartite graph. We select the adjacency matrix, where the rows represent source vertices and columns represent destination vertices. Three further properties specify the studied complete bipartite graph:

1. each source vertex is connected with each destination vertex by an edge,
2. each edge is colored by exactly one of the four colors,
3. it is not allowed, that all edges of any quadruple of edges

$$\{e(s_1,d_1),e(s_1,d_2),e(s_2,d_1),e(s_2,d_2)\}$$

are colored by the same color.

These three conditions mean in terms of the used data structure that all positions of the adjacency matrix must be colored by one of the four colors, and it is not allowed that in the four cross points of any pair of rows and any pair of columns the same color appears. It should be noted that due to the third property mentioned above a tight relationship to bipartite Ramsey numbers [7] exists. A comprehensive theory for such a grid coloring with regard to several fixed numbers of colors is published in [5].

Until now it was unknown whether there is a rectangle-free 4-colorable grid of the size 17×17, 17×18, 18×17, or 18×18. When the complexity of the problem will be considered, then the grid $G_{18,18}$ has $18*18=324$ positions, and one of the four colors must be selected for each of them. Hence, there are $4^{324}=1.16798*10^{195}$ different patterns in which one of the four colors is assigned to each of the 18×18 positions of the grid. Assume we are able to evaluate one pattern in one nano-second (10^{-9} seconds) and we spend 100 years ($3*10^9$ seconds) then we must repeat the job $3.8*10^{176}$ times in order to know whether there is an allowed color pattern and if YES which valid color pattern exists. So we see that we are going to solve an extremely complex problem. Because a problem of such a complexity never has been solved before, we describe the solution more in detail. These details can be adopted to teach students approaches to solve similar extremely hard problem. The complexity also shows that a simple computer program will not do the job; many additional considerations are necessary which require deep knowledge in the fields of computer science and mathematics.

8. Binary Model of Rectangle-Free 4-Colorable Grids

The four colors can be represented by the four values $1,2,3,4$. Internally the computer works with binary values. Hence, we have to map the four-color value into the Boolean space. The four-color values of a single grid element x can be expressed by two Boolean values a and b. Table 8 shows the used mapping. Function (28) depends on eight Boolean variables and has a Boolean result that is true in the case that the colors in the corners of the rectangle selected by the rows

Table 8. Mapping of the 4-valued color x to 2 Boolean variables a and b

x	a	b
1	0	0
2	1	0
3	0	1
4	1	1

r_i and r_j and by the columns c_k and c_l are equal to each other. Each conjunction in this function represents the rectangle condition for one color. The index *ecb* of Function (28) means *equal color binary*.

$$\begin{aligned} f_{ecb}(a_{r_i,c_k}, b_{r_i,c_k}, a_{r_i,c_l}, b_{r_i,c_l}, a_{r_j,c_k}, b_{r_j,c_k}, a_{r_j,c_l}, b_{r_j,c_l}) = \\ (\overline{a}_{r_i,c_k} \wedge \overline{b}_{r_i,c_k} \wedge \overline{a}_{r_i,c_l} \wedge \overline{b}_{r_i,c_l} \wedge \overline{a}_{r_j,c_k} \wedge \overline{b}_{r_j,c_k} \wedge \overline{a}_{r_j,c_l} \wedge \overline{b}_{r_j,c_l}) \vee \\ (a_{r_i,c_k} \wedge \overline{b}_{r_i,c_k} \wedge a_{r_i,c_l} \wedge \overline{b}_{r_i,c_l} \wedge a_{r_j,c_k} \wedge \overline{b}_{r_j,c_k} \wedge a_{r_j,c_l} \wedge \overline{b}_{r_j,c_l}) \vee \\ (\overline{a}_{r_i,c_k} \wedge b_{r_i,c_k} \wedge \overline{a}_{r_i,c_l} \wedge b_{r_i,c_l} \wedge \overline{a}_{r_j,c_k} \wedge b_{r_j,c_k} \wedge \overline{a}_{r_j,c_l} \wedge b_{r_j,c_l}) \vee \\ (a_{r_i,c_k} \wedge b_{r_i,c_k} \wedge a_{r_i,c_l} \wedge b_{r_i,c_l} \wedge a_{r_j,c_k} \wedge b_{r_j,c_k} \wedge a_{r_j,c_l} \wedge b_{r_j,c_l}) \end{aligned} \tag{28}$$

The conditions of the rectangle-free 4-color problem on a grid $G_{m,n}$ are achieved when Function f_{ecb} (28) is equal to 0 for all rectangles which can be expressed by:

$$\bigvee_{i=1}^{m-1} \bigvee_{j=i+1}^{m} \bigvee_{k=1}^{n-1} \bigvee_{l=k+1}^{n} f_{ecb}(a_{r_i,c_k}, b_{r_i,c_k}, a_{r_i,c_l}, b_{r_i,c_l}, a_{r_j,c_k}, b_{r_j,c_k}, a_{r_j,c_l}, b_{r_j,c_l}) = 0 \,. \tag{29}$$

It is very important to know that this equation is a *correct model*. Each solution of this equation is a solution of the problem. From the solution of Equation (29) we know immediately whether there is no solution, one solution or more solutions. However, it is extremely hard to solve Equation (29) for $m = 18$ rows and $n = 18$ columns.

9. Basic Approaches and Results

9.1. Solving Boolean Equations

In order to solve the highly complex coloring problem we need deep knowledge of its properties. Very often you can find the opinion that it is only necessary to have a good computer and a good programmer, and then the rest is done without any further problems. Here it is easy to see that this opinion only holds up to a given size of the problem. The complexity $2^{648} \approx 10^{195}$ is far beyond imagination to yourself, and it is a good reason to stop any further approaches to the problem. On the other side the constructive solution of such a problem eliminates the reasoning that problems with exponential complexity are considered as intractable.

The Boolean equation (29) can be completely solved for small grid sizes. The representation by ternary vectors of XBOOLE [10, 14, 18] helps to restrict the required memory. Table 9 shows the detailed results for the grids of two to seven rows (labeled by r) and of two columns (labeled by c). The column labeled by v gives the number of Boolean variables of Equation (29). The

Table 9. Solutions of the Boolean equation (29)

r	c	v	TV	solutions	forbidden	ratio
2	2	8	24	252	4	1.56
3	2	12	304	3,912	184	4.49
4	2	16	3,416	59,928	5,608	8.56
5	2	20	36,736	906,912	141,664	13.51
6	2	24	387,840	13,571,712	3,205,504	19.11
7	2	28	4,061,824	201,014,784	67,420,672	25.12

column labeled by TV enumerates the number of ternary vectors needed to express the solutions given in the next column. The benefit of the ternary representation is obvious. Because XBOOLE uses orthogonal ternary vectors it is easy to calculate the number of all solutions.

The number of all possible color patterns is defined by 4^{r*c} which is equal to 2^v, the power of 2 to the number of variables v. The difference between 2^v and the number of solutions is equal to the number of forbidden color patterns. The column ratio in Table 9 gives the percentage of the number of forbidden patterns divided by all 2^v possible color patterns.

For the simplest grid $G_{2,2}$ almost all of the $2^8 = 256$ color patterns are correct solutions. Only the four patterns specified by Function (28) are forbidden. The forbidden fraction of 4-color patterns of $G_{2,2}$ is only 1.56%. This ratio grows to more than 25% for the grid $G_{7,2}$. That means that the number of forbidden patterns grows stronger than the number all possible 4-color patterns by enlarging the size of the grid. From this observation we learn that there must be a 4-colorable grid $G_{m,n}$ with the property that at least one of the grids $G_{m+1,n}$ or $G_{m,n+1}$ is not rectangle-free 4-colorable. The practical results of Table 9 confirm the theory of [5] that forbidden color patterns grow even stronger than the exponential growth of the number of possible color patterns for growing numbers of rows and columns of the grid.

9.2. Exploit Permutations of Colors, Rows, and Columns

The limit in the previous approach was the required memory of about 800 Megabytes to represent the solution of the grid $G_{7,2}$ which could be calculated in less than 5 seconds. To break the limitation of memory requirements we exploited some heuristic properties of the problem:

1. Knowing one single solution of the 4-color problem, 4! = 24 permutations of this solution with regard to the four colors are also solutions.

2. The permutation of rows and columns of a given solution pattern creates another pattern that is a valid solution, too.

3. A nearly uniformly distribution of the colors in both the rows and the columns is given for the largest number of rectangle-free 4-colored grids.

Hence, in [22] we applied a heuristic which restricts to the calculation of such solutions having a single fixed uniform distribution of the colors in the top row and in the left column. We restrict in this experiment the calculation to 12 columns and 2 Gigabytes of available memory.

Table 10. Selected solutions of the Boolean equation (29) with fixed uniform distribution of the colors in the top row and in the left column

r	c	v	TV	solutions
2	2	8	1	4
2	12	48	6,912	2,361,960
3	2	12	1	16
3	8	48	4,616,388	4,616,388
4	2	16	1	64
4	5	40	674,264	12,870,096
5	2	20	1	256
5	4	40	573,508	12,870,096
6	2	24	4	960
6	3	36	15,928	797,544
7	2	28	16	3,600
7	3	42	183,152	104,93,136
8	2	32	64	3,600
8	3	48	2,152,819	136,603,152
9	2	36	256	50,625
10	2	40	768	182,250
15	2	60	147,456	104,162,436
19	2	76	6,553,600	14,999,390,784

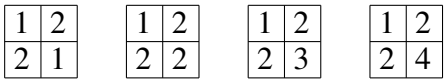

Figure 7. The selected rectangle-free 4-colored grids $G_{2,2}$ of the first row in Table 10.

Table 10 shows selected results of the explained restricted calculation repeated from [22]. Figure 7 depicts the four selected rectangle-free 4-colored grids of the first row in Table 10. Using the same memory size, the number of Boolean variables could be enlarged from 28 for $G_{7,2}$ to 76 for $G_{19,2}$. That means, by utilizing properties of the 4-color problem mentioned above, we have solved a problem that is $2^{48} = 281,474,976,710,656$ times larger than before, but again the available memory size restricts the solution of larger rectangle-free 4-colorable grids [22].

9.3. Exchange of Space and Time

Function (28) describes the forbidden patterns of a single rectangle. All rectangles are specified by all possible pairs of rows and all possible pairs of columns. Hence, there are

$$n_r(m,n) = \binom{m}{2} * \binom{n}{2} \tag{30}$$

rectangles labeled by `all_rect` in the following program fragments. The conditions of these

```
for(i = 0; i < all_rect; i++)
    aps = DIF(aps, f_ecb[i]);
```

(a)

```
for(i = 0; i < all_rect; i++) {
    aps = DIF(aps, f_ecb[i]);
    if(NTV(aps) > SplitLimit) {
        SPLIT(aps, aps0, aps1);
        aps_stack.PUSH(aps1);
        position_stack.PUSH(i);
        aps = aps0;
    }
    if(NTV(aps) == 0) {
        aps = aps_stack.POP();
        i = position_stack.POP();
    }
}
```

(b)

Figure 8. Iterative approach: (a) with unrestricted space requirements, (b) with restricted space requirements.

rectangles must be excluded from all patterns of the Boolean space B^{2*m*n} which can be calculated using the `DIF`-operation of XBOOLE [10, 14, 18].

The XBOOLE-operation `C=DIF(A,B)` calculates the *set difference* $C = A \setminus B = A \cap \neg B$ for the given TVLs `A` and `B` such that as much as possible dash elements of `A` remain in the result `C`. We use the DIF-operation of XBOOLE to solve the grid coloring problem. Assume `aps` is the *actual partial solution* which is initialized by the whole Boolean space B^{2*m*n}; then the core algorithm of Figure 8 (a) solves the rectangle-free 4-coloring problem when unrestricted memory space can be used.

It is an important drawback of the approach of Figure 8 (a) that the size of `aps` typically increases extremely up to a certain index i and decreases later on. Therefore, we implemented another iterative algorithm that allows us to exchange space and time.

If the extremely large size of `aps` avoids the next iteration step we split `aps` into `aps0` and `aps1`, solve both sub-problems sequentially, and combine both solutions at the end. This approach can be repeatedly utilized as shown in Figure 8 (b). In this algorithm we use additionally the `NTV`-operation of XBOOLE. The `NTV`-operation returns the *number of ternary vectors* of the given TVL. The `SPLIT`-operation splits the given TVL `aps` approximately in the middle into the TVLs `aps0` and `aps1`. Two stacks with the access operations `PUSH` and `POP` are used as storage for the recursive calculation.

We applied the algorithm of Figure 8 (b) to grids of 12 rows, used a fixed value `SplitLimit = 400`, and canceled the calculation when the first solutions were found.

The approach of exchanging space and time allows us again an extreme improvement: solutions for rectangle-free 4-colored grids which are modeled with up to 384 Boolean variables were found instead of 76 variables in the second (already improved) approach. This means that the approach of exchanging space and time for the 4-color grid problems allows us solving problems which are $2^{308} = 5.214812 * 10^{92}$ times larger than before [22].

Table 11. Sequential solution using the algorithm of Figure 8 (b) canceled in the case of a non-empty solution set

rows	columns	variables	ternary vectors	solutions	maximal stack size
12	2	48	337	6,620	3
12	3	72	147	2,423	22
12	4	96	319	7,386	30
12	5	120	236	1,188	47
12	6	144	181	1,040	61
12	7	168	231	627	69
12	8	192	109	413	81
12	9	216	72	227	79
12	10	240	34	103	87
12	11	264	40	109	88
12	12	288	112	293	82
12	13	312	51	81	81
12	14	336	82	1,415	97
12	15	360	1	1	80
12	16	384	2	3	81

The time to solve the 4-color grids of Table 11 were less than 1 second until 12 columns, less than 3 seconds until 15 columns, and grows to 427 seconds for 16 columns. An approach to reduce the required time is given by parallel computing [24].

10. Steps to Solve all Rectangle-free 4-colored Grids up to the Size 18×18

10.1. Applying SAT-solvers

The power of SAT-solvers [2] has been improved over the last decades forced by several SAT-competitions. We tried to solve the rectangle-free 4-color grid problem using the best SAT-solvers from the SAT-competitions of the last years. Equation (29) can be easily transformed into a SAT-equation by negation of both sides and the application of De Morgan's law to the Boolean expression on the left-hand side. In this way we get the required conjunctive form for the SAT-solver (31).

$$\bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} \bigwedge_{k=1}^{n-1} \bigwedge_{l=k+1}^{n} \overline{f_{ecb}(a_{r_i,c_k}, b_{r_i,c_k}, a_{r_i,c_l}, b_{r_i,c_l}, a_{r_j,c_k}, b_{r_j,c_k}, a_{r_j,c_l}, b_{r_j,c_l})} = 1 \ . \tag{31}$$

Table 12 shows the required time to find the first solution for square rectangle-free 4-colored grids $G_{12,12}$, $G_{13,13}$, $G_{14,14}$, and $G_{15,15}$ using the SAT-solver *clasp* [6], *lingeling* [1], *plingeling* [1], and *precosat* [1]. Figure 9 shows the 4-colored grid $G_{15,15}$ found by clasp within 46 and a half minutes.

From the utilization of the SAT-solvers we learned that:

Table 12. Time to solve square rectangle-free 4-colored grids using different SAT-solvers

rows columns	variables	time in minutes:seconds clasp	lingeling	plingeling	precosat
12	288	0:00.196	0:00.900	0:00.990	0:00.368
13	338	0:00.326	0:01.335	0:04.642	0:00.578
14	392	0:00.559	0:03.940	0:02.073	0:00.578
15	450	46:30.716	54:02.304	73:05.210	120:51.739

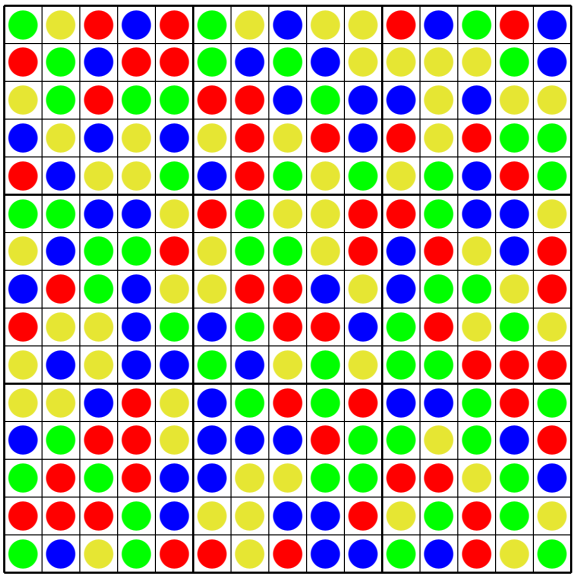

Figure 9. Rectangle-free 4-colored grid $G_{15,15}$ found by the SAT-solver *clasp* in about 46.5 minutes

1. SAT-solvers are powerful tools that are able to solve rectangle-free 4-colored grid up to $G_{15,15}$,
2. it is not possible to calculate a rectangle-free 4-colored grid larger than $G_{15,15}$ directly.

The reasons for the second statement are first that the search space for the 4-colored grid $G_{16,16}$ is $2^{62} = 4.61 * 10^{18}$ times larger than the search space for the 4-colored grid $G_{15,15}$, and second that the fraction of the 4-colorable grids is reduced for the larger grid even stronger. We take as base of the time measurement the age of our planet which is about 4 billion years ($4 * 10^9$ years). Based on the measured time to find the first rectangle-free 4-colored grid $G_{15,15}$ and knowing the larger search space it can be estimated that it takes approximately 10,000 times the age of the Earth to find a rectangle-free 4-colored grid $G_{16,16}$.

10.2. Construction of Well-structured Rectangle-free 4-colored Grids $G_{16,16}$ and $G_{16,20}$

The found solution for $G_{15,15}$ of Figure 9 gave us a hint how larger 4-colored grids may be constructed. The evaluation of the rows and columns over the whole grid shows that the colors

are nearly uniformly distributed. However, single colors dominate in the sub-grids. Due to the 16 positions in rows and columns of $G_{16,16}$ and the four allowed colors the sub-grids $G_{4,4}$ are taken into account. From our complete solutions for small grids we know that maximal 9 positions can be occupied by a single color without violation of the rectangle-free condition.

Such maximal 1-colored $G_{4,4}$ grids cannot be repeated in rows or columns, but in diagonal order. In this way, a $G_{8,8}$ grid can be dominated by two colors. In such a structure both red-dominated $G_{4,4}$ sub-grids can be extended by a single position of color green. Vice versa both green-dominated $G_{4,4}$ sub-grids can be extended by a single position of color red. It remain $8*8-4*9-4*1=24$ positions which can be filled up with the colors blue and yellow, respectively. An analog $G_{8,8}$ grid dominated by the colors blue and yellow can be built. Similarly, the whole 4-colored grid $G_{16,16}$ can be constructed using reversed 2-color dominated $G_{8,8}$ grids. Figure 10 shows the constructed 4-colored grid $G_{16,16}$ [17].

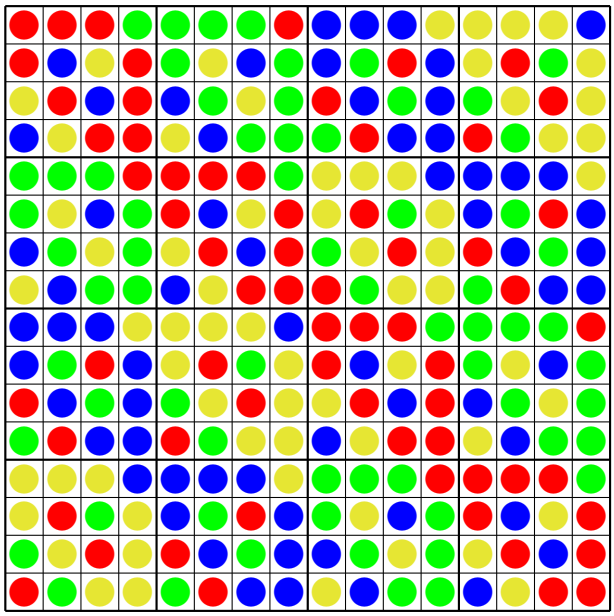

Figure 10. Rectangle-free 4-colored grid $G_{16,16}$ constructed using maximal 1-colored sub-grids $G_{4,4}$

In the sub-grids $G_{4,4}$ of Figure 10 each pair of rows and each pair of columns is covered by the dominating color. Hence, in these ranges each color is only allowed in a single position. Cyclically shifted combinations of all four colors allow the extension of the 4-colored grid $G_{16,16}$ of Figure 10 to the rectangle-free 4-colored grid $G_{16,20}$ as shown in Figure 11 [17].

Similarly to the 4-colored grid $G_{16,20}$ of Figure 11 a rectangle-free 4-colored grid $G_{20,16}$ can be constructed based on the rectangle-free 4-colored grid $G_{16,16}$ of Figure 10. Unfortunately, it is not possible to build a rectangle-free 4-colored grid $G_{17,17}$ that includes the 4-colored grid $G_{16,16}$ of Figure 10.

10.3. Restriction to a Single Color of Rectangle-free 4-colored Grids

Due to the high complexity, a divide and conquer approach may facilitate the solution of the rectangle-free 4-colored grid $G_{17,17}$ or even the grid $G_{18,18}$. The divide step restricts first to

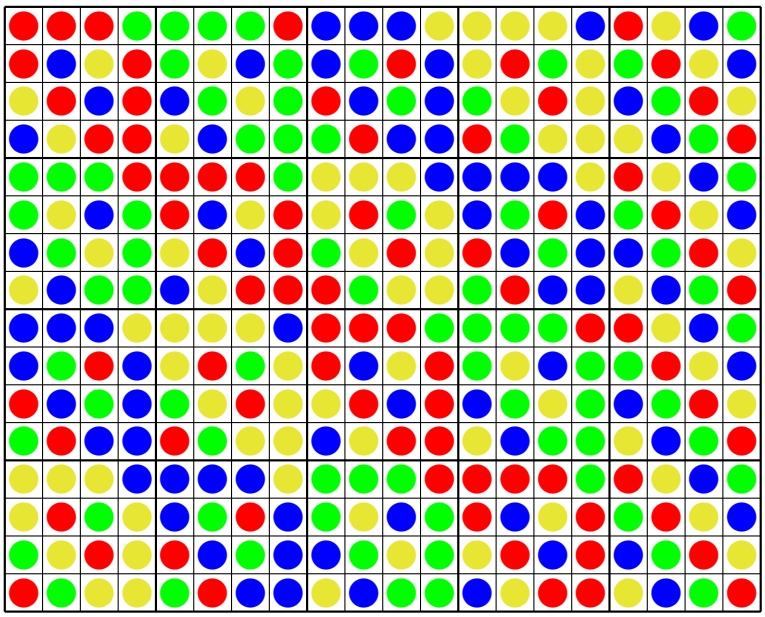

Figure 11. Rectangle-free 4-colored grid $G_{16,20}$ constructed using maximal 1-colored sub-grids $G_{4,4}$ and cyclic extension of all four colors

one single color. Due to the pigeonhole principle, at least one fourth of the grid positions must be covered by the first color without contradiction to the color restrictions. When such a partial solution is known, the same fill-up step must be executed taking into account the already fixed positions of the grid. This procedure must be repeated for all four colors.

The advantage of this approach is that a single Boolean variable describes whether the color is assigned to a grid position or not. Such a restriction to one half of the needed Boolean variables reduces the search space form $2^{2*18*18} = 1.16 * 10^{195}$ to $2^{18*18} = 3.41 * 10^{97}$ for the grid $G_{18,18}$ drastically.

Function f_{ecb} (28) which describes equal colors in the corners of a rectangle can be simplified to f_{ecb1} (32) for a single color in the divide and conquer approach:

$$f_{ecb1}(a_{r_i,c_k}, a_{r_i,c_l}, a_{r_j,c_k}, a_{r_j,c_l}) = (a_{r_i,c_k} \wedge a_{r_i,c_l} \wedge a_{r_j,c_k} \wedge a_{r_j,c_l}) \; . \tag{32}$$

By transformation into a SAT-problem we get:

$$\bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} \bigwedge_{k=1}^{n-1} \bigwedge_{l=k+1}^{n} \overline{f_{ecb1}(a_{r_i,c_k}, a_{r_i,c_l}, a_{r_j,c_k}, a_{r_j,c_l})} = 1 \; . \tag{33}$$

A disadvantage of this approach is that the implicit assignment of exactly one color to each grid position is lost. The values of the pair of variables $(a_{r_i,c_k}, b_{r_i,c_k})$ in the solution of (31) determine one of the four colors for the position of the row r_i and the column c_k. The value of the single variable a_{r_i,c_k} in the solution of (33) determines only whether the chosen color is assigned, $a_{r_i,c_k} = 1$, or one of the remaining colors must be used, $a_{r_i,c_k} = 0$.

The equation $f_{ecb1} = 0$ has 15 solutions; only the assignment of values 1 to all a-variables is excluded. One solution of $\overline{f}_{ecb1} = 1$ calculated by a SAT-solver will be the assignment of values

1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
0	1	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0
0	0	1	0	1	0	0	0	1	0	0	0	0	0	1	1	0	0
0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0
0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1
0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0
0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0

Figure 12. Rectangle-free grid $G_{18,18}$ colored by one fourth of all positions with the first color (1).

0 to all a-variables. This is a correct solution; the chosen color does not conflict with the color restriction when it is not assigned to any grid position. However, we are not interested in this trivial solution; we are looking for a solution where the chosen color covers one fourth of the grid positions. Therefore, a SAT-solver cannot solve this problem directly.

We developed a quite complicated algorithm that allows us to find solutions of (33) with maximal assignments of values 1. For the sake of space, we must exclude the details of this approach from this paper. These details are the topic of the paper [21] and summarized in Section 1.5 of [13]. However, the results are important for the following final step to the solution. Using XBOOLE, we developed a program that implements the mentioned approach and calculated a rectangle-free assignment of 81 values 1 to the 324 grid positions of $G_{18,18}$. Figure 12 shows this single color solution.

Our effort to fill up the rectangle-free 1-colored grid $G_{18,18}$ of Figure 12 with the second color on again 81 grid positions failed. This results from the fact that the freedom for the choice of the positions is restricted by the assignments of the first color. We learned from this approach that it is not enough to know a correct coloring for one color, these assignments must not constrain the assignment of the other colors.

10.4. Cyclic Color Assignments of Rectangle-free 4-colored Grids

The smallest restrictions for the coloring of a grid by the four colors are given when the number of assignments to the grid positions is equal for all four colors. For square grids $G_{m,n}$, $m = n$, with an even number of m rows and n columns, quadruples of all grid positions can be chosen which contain all four colors. There are several possibilities of such selections of quadruples. One of them is cyclic rotation of a chosen grid position by 90 degrees around the center of the grid.

r_1	s_1	t_1	r_2
t_4	u_1	u_2	s_2
s_4	u_4	u_3	t_2
r_4	t_3	s_3	r_3

Figure 13. Cyclic quadruple in a grid $G_{4,4}$.

Figure 13 illustrates this possibility for the simple grid $G_{4,4}$. The quadruples are labeled by the letters r,s,t, and u. The attached index specifies the element of the quadruple.

In addition to the color restriction (33) for the chosen single color we can require that this color occurs exactly once in each quadruple. This property can be expressed by two additional rules. For the corners of the grid of Figure 13, for instance, we model as the first rule the requirement:

$$r_1 \vee r_2 \vee r_3 \vee r_4 = 1 \ , \tag{34}$$

so that at least one variable r_i must be equal to 1. As the second rule, the additional restriction

$$\begin{aligned} &(r_1 \wedge r_2) \vee (r_1 \wedge r_3) \vee (r_1 \wedge r_4) \vee \\ &(r_2 \wedge r_3) \vee (r_2 \wedge r_4) \vee (r_3 \wedge r_4) = 0 \end{aligned} \tag{35}$$

prohibits that more than one variable r_i is equal to 1.

A SAT-formula can be constructed using (33) and for all cyclic quadruples as illustrated in Figure 13 both the fitted requirements (34) and the fitted restrictions (35) negated using De Morgan's laws. The solution of such a SAT-formula for a square grid of even numbers of rows and columns must assign exactly one fourth of the variables to 1. Such a solution can be used rotated by 90 degrees for the second color, rotated by 180 degrees for the third color, and rotated by 270 degrees for the forth color without any contradiction.

We generated the cnf-file [3] of this SAT-formula which depends on 324 variables and contains 23,976 clauses for the grid $G_{18,18}$ and tried to find a solution using the SAT-solver *clasp* [6]. The SAT-solver *clasp* found the first cyclic reusable solution for the grid $G_{18,18}$ after 2 days 10 hours 58 minutes 21.503 seconds. Figure 14 shows this solution for the first color of the grid $G_{18,18}$.

Using the core solution of Figure 14 we have constructed the rectangle-free 4-colored grid $G_{18,18}$ of Figure 15 by three times rotating around the grid center by 90 degrees each and assigning the next color [13, 17].

Many other rectangle-free 4-colored grids can be created from the solution in Figure 15 by permutations of rows, columns and colors. Several rectangle-free 4-colored grids $G_{17,18}$ originate from the rectangle-free 4-colored grid $G_{18,18}$; by removing any single row or column we get the rectangle-free 4-colored grids $G_{17,18}$ or $G_{18,17}$. Obviously, several so far unknown rectangle-free 4-colored grids $G_{17,17}$ can be selected from the rectangle-free 4-colored grid $G_{18,18}$ of Figure 15 removing both any single row and any single column.

It should be mentioned that the approach of cyclic reusable single assignments can be applied to the 4-colored square grids of an odd number of rows and columns, too. The central position must be colored with the first chosen color. Figure 16 shows the principle of the quadruple assignment in this case.

1	0	0	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0	1	0
1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1
0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1
0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0
0	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	1	0	0
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0	0	1	1	0	1	1	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0
1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
0	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0

Figure 14. Cyclic reusable rectangle-free coloring of the grid $G_{18,18}$.

The SAT-solver *clasp* found the first cyclic 4-colorable solution for odd grids up to $G_{15,15}$ in less than 0.6 seconds but could not solve this task for the grid $G_{17,17}$ within two months.

11. Experiences and new Challenges for Teaching in the Boolean Domain

Teaching in the Boolean domain requires a comprehensive theoretic basis. These topics comprise areas of the *discrete mathematics* such as algebraic structures and their properties and especially the *Boolean algebra*. Students must know the Boolean operations and transformation rules of Boolean expressions. An appropriate text book for these studies is [10]. This book starts from the scratch, provides the required theoretical basis, and extends this basis by the *Boolean Differential Calculus* (BDC) to study changes in Boolean systems.

Motivated by the permanent extension of applications, digital circuits are one of main teaching topics in the Boolean domain. Additionally, more general logic applications help the students to realize the vital importance of comprehensive knowledge in the Boolean domain. Our text book [10] comprises general logic and arithmetic as well as digital systems. The introduced operations of the BDC allow us to present the most powerful decomposition methods which are missing in other text books.

It is state of the art to teach special methods for analysis and design of digital circuits, starting with combinatorial circuits followed by sequential circuits for finite state machines. The exponential complexity of Boolean functions and the limited time for exercises restrict the task to be solved to very small examples of very few Boolean variables. Such examples are far away from real practical applications. Despite the small number of Boolean variables, the students will notice the limited abilities of human beings to manipulate the exponential number of Boolean values without any mistake and they will realize that computers with convenient software are needed.

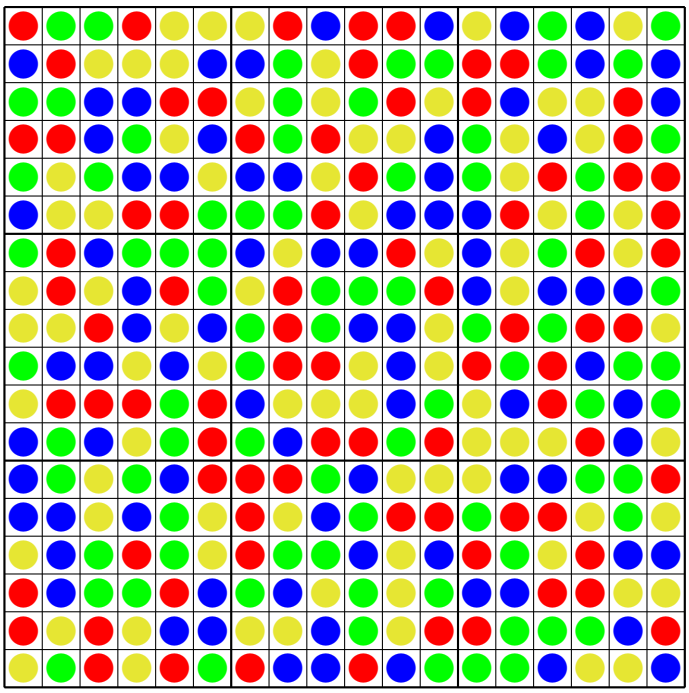

Figure 15. Rectangle-free 4-colored grid $G_{18,18}$.

r_1	s_1	t_1	u_1	r_2
u_4	v_1	w_1	v_2	s_2
t_4	w_4	x_1	w_2	t_2
s_4	v_4	w_3	v_3	u_2
r_4	u_3	t_3	s_3	r_3

Figure 16. Cyclic quadruple in a grid $G_{5,5}$.

The software brings us into the field of computer science. Many details in programming sidetracks the students from the key problems in the Boolean domain. We weaken this problem by the development of the XBOOLE system [14, 18]. As explained above, sets of binary vectors or Boolean functions are represented in XBOOLE by *ternary vector lists* (TVLs). In this way the bit-level of programming languages is raised to the level of models in the Boolean domain. Both the ternary representation and the system of many Boolean space with unlimited Boolean variables weaken the complexity problem.

In order to skip programming skills as much as possible from the teaching process in the Boolean domain, we developed the XBOOLE-Monitor. The XBOOLE-Monitor works like a Boolean pocket-calculator. All set operations (which are isomorph to the Boolean operations) and the derivative operations of the BDC can be executed for sets or Boolean functions represented as TVL. Many further operations extent the field of applications. An includes help system explains both the basic concepts and all provided operations. As mentioned above, the XBOOLE-Monitor can be downloaded (for free) at

`http://www.informatik.tu-freiberg.de/xboole/` .

We have very good experiences in the application of the XBOOLE-Monitor both in lectures and exercises of courses about digital systems. Because all manipulations on the bit level are correctly executed by XBOOLE on the bit level, the teaching process can be focused on the level of algorithms. It is an important benefit for the cognition process of the students that the developed algorithms do not remain as abstract entity but can immediately transformed into an sequence of XBOOLE-operations and stored as a text file. We call such a sequence of XBOOLE-operations problem program (PRP). The XBOOLE-Monitor is able to execute such PRPs so that the solution of the concrete problem is found and can be verified by the students. The very short time from an idea across the the algorithm to the final solution of the problem using the XBOOLE-Monitor allows us to include this method of deep insight directly in our lectures.

The students should engross this method in exercises. Our book [18] contains many tasks of several fields of applications which can be solved using the mentioned XBOOLE-Monitor. This book supports the readers (often students) by means of both associated solutions (enumerated at the end of the same book) and hints to the needed theory given in [10]. Our undergraduate students successfully use the XBOOLE-Monitor in courses about digital systems.

A key issue of this approach: the students must find an appropriate *correct Boolean model*. This starts with the definition of the needed variables. In the case of multiple-valued problems the binary encoding of multiple-valued variables is necessary. As shown above, the properties of the problem have strong influence to a convenient encoding. Due to the need of the specification of intermediate solution sets, the 1-of-n encoding fits well in the case of Sudoku. The most compact binary encoding restricts the search space in the case of the coloring of large grids. The detection of both the requirements and the conditions enables the specification of systems of Boolean equations as a very universal solution approach.

The XBOOLE-Monitor utilizes the underlying XBOOLE-Library. This software library comprises more than hundred functions which can be used in the programming languages C and C++. Our students on the master level solve more challenging tasks using the XBOOLE-Library. As demonstrated in the very complex grid coloring, the students must learn that knowledge of different fields of science must be combined to solve so far open problems. Additionally, the different skills of human beings and computers must be utilized together. Skills in programming are also needed to prepare SAT-equations of several thousands of clauses. The analysis of intermediate results must be recognized as important source for future steps to the solution.

It can be seen that the so-called exponential complexity is not really an argument to put a problem aside. There are problems where no mathematical proof can be found, but the utilization of all (including very hidden) properties, correct algorithms and convenient software enables the final solution.

Do to the wide field of applications the concepts of Boolean equations together with the respective solution methods and the respective software must be a core concept of many different courses.

12. Conclusion

We explored in this paper some combinatorial problems with increasing complexity. The highlights were the coloring of the grids $G_{17,17}$, $G_{17,18}$, $G_{18,17}$, $G_{18,18}$. Our study has shown that the fraction of rectangle-free 4-colorable grids of the size 18×18 is extremely small. Hence, finding

a rectangle-free 4-colored grid $G_{18,18}$ out of the unimaginably large number of $1.16798 * 10^{195}$ possible assignments of 4 colors is significantly more difficult than detecting a single atom within the whole universe. We successfully verified the presented general approach in our solution of the last open, even more complicated rectangle-free 4-colored grid $G_{12,21}$ [20].

We evaluated several approaches, developed and utilized different computer programs, and combined all the collected knowledge. From all our approaches we learned, neither a fitting default computer program nor a human being will be able to solve such a highly complex problem, but commonly we were able to find rectangle-free 4-colored solutions for the grid of the size 18×18 and consequently for all sub-grids.

It can also be seen that the so-called *exponential complexity* is not really an argument to put a problem aside. It needs careful considerations how important the solution of this problem will be in order to justify the solution efforts. It is also very necessary to think carefully about the organization of the required *interdisciplinary cooperation*. *Mathematics* is more applicable than ever before, but only by an *effective cooperation* with fields of *Computer Science*!

A very general approach to solve many different finite discrete problems are their modeling by Boolean equations. Available software, like the XBOOLE-Monitor, allows everybody to calculate the needed solution. The inclusion of this approach in many courses may help to move the focus from less important details to key issues. This approach can be generalized for high-level teaching to solve extremely complex problems.

References

[1] Biere, A. (2010). *Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010*. (10/1). Institute for Formal Models and Verification, Kepler University.

[2] Biere, A., Heule, M. J. H., Maaren, H. van, and Walsh, T., eds. (2009). *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press.

[3] DIMACS. (1993). *Satisfiability Suggested Format*. 1–8. `http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf`.

[4] Felgenhauer, B. and Jarvis, F. (2005). *Enumerating possible Sudoku grids*. 1–7. `http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf`.

[5] Fenner, S., Gasarch, W., Glover, C., and Purewal, S. (2009). *Rectangle Free Coloring of Grids*. URL: `http://www.cs.umd.edu/~gasarch/papers/grid.pdf`.

[6] Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. (2007). clasp: A Conflict-Driven Answer Set Solver. *9th International Conference on Logic Programming and Nonmonotonic Reasoning*. Vol. LNAI 4483. LPNMR (pp. 260–265). Springer. Tempe, AZ, USA.

[7] Graham, R. L., Rothschild, B. L., and Spencer, J. H. (1990). *Ramsey Theory - Second Edition*. New York: John Wiley & Sons.

[8] Lynce, I. and Ouaknine, J. (2006). Sudoku as a SAT Problem. *9th International Symposium on Artificial Intelligence and Mathematics*.

[9] Marx, D. (2004). Graph Coloring Problems and Their Applications in Scheduling. *Periodica Polytechnica, Electrical Engineering* **48**(1): 11–16.

[10] Posthoff, C. and Steinbach, B. (2004). *Logic Functions and Equations - Binary Models for Computer Science*. Dordrecht, The Netherlands: Springer.

[11] Posthoff, C. and Steinbach, B. (2010). The Solution of Discrete Constraint Problems Using Boolean Models. *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence – ICAART 2010*. Filipe, J. and Fred, A., eds. ICAART (pp. 487–493). Valencia, Spain.

[12] Posthoff, C. and Steinbach, B. (2011). The Solution of SAT Problems Using Ternary Vectors and Parallel Processing. *International Journal of Electronics and Telecommunications (JET)* **57**(3): 233–249.

[13] Steinbach, B., ed. (2014). *Recent Progress in the Boolean Domain*. Newcastle upon Tyne, UK: Cambridge Scholars Publishing.

[14] Steinbach, B. (1992). XBOOLE - A Toolbox for Modelling, Simulation, and Analysis of Large Digital Systems. *System Analysis and Modeling Simulation* **9**(4): 297–312.

[15] Steinbach, B. and Posthoff, C. (2013). Artificial Intelligence and Creativity - Two Requirements to Solve an Extremely Complex Coloring Problem. *Proceedings of the 5th International Conference on Agents and Artificial Intelligence*. Filipe, J. and Fred, A., eds. Vol. 2. ICAART (pp. 411–418). Barcelona, Spain.

[16] Steinbach, B. and Posthoff, C. (2013). *Boolean Differential Equations*. Morgan & Claypool Publishers.

[17] Steinbach, B. and Posthoff, C. (2012). Extremely Complex 4-Colored Rectangle-Free Grids: Solution of Open Multiple-Valued Problems. *Proceedings of the IEEE 42nd International Symposium on Multiple-Valued Logic*. ISMVL (pp. 37–44). Victoria, BC, Canada. DOI: 10.1109/ISMVL.2012.12.

[18] Steinbach, B. and Posthoff, C. (2009). *Logic Functions and Equations - Examples and Exercises*. Springer Science + Business Media B.V.

[19] Steinbach, B. and Posthoff, C. (2014). Multiple-Valued Problem Solvers – Comparison of Several Approaches. *Proceedings of the IEEE 44th International Symposium on Multiple-Valued Logic (ISMVL 2014)*. (pp. 25–31). Bremen, Germany. DOI: 10.1109/ISMVL.2014.13.

[20] Steinbach, B. and Posthoff, C. (2013). Solution of the Last Open Four-Colored Rectangle-free Grid - an Extremely Complex Multiple-Valued Problem. *Proceedings of the IEEE 43rd International Symposium on Multiple-Valued Logic (ISMVL 2013)*. (pp. 302–309). Toyama, Japan. DOI: 10.1109/ISMVL.2013.51.

[21] Steinbach, B. and Posthoff, C. (2012). Utilization of Permutation Classes for Solving Extremely Complex 4-Colorable Rectangle-Free Grids. *Proceedings of the IEEE 2012 International Conference on Systems and Informatics (ICSAI 2012)*. (pp. 2361–2370). Yantai, China.

[22] Steinbach, B., Posthoff, C., and Wessely, W. (2010). Approaches to Shift the Complexity Limitations of Boolean Problems. *Proceedings of the Seventh International Conference on Computer Aided Design of Discrete Devices*. CAD DD (pp. 84–91). Minsk, Belarus.

[23] Steinbach, B. and Werner, M. (2014). XBOOLE-CUDA - Fast Boolean Operations on the GPU. *11th International Workshop on Boolean Problems*. IWSBP (pp. 75–84). Freiberg, Germany.

[24] Steinbach, B., Wessely, W., and Posthoff, C. (2010). Several Approaches to Parallel Computing in the Boolean Domain. *1st International Conference on Parallel, Distributed and Grid Computing*. PDGC (pp. 6–11). Jaypee University of Information Technology Waknaghat, Solan, H.P., India.

[25] Weber, T. (2005). *A SAT-based Sudoku Solver*. TU Munich.