

THE SOLUTION OF DISCRETE CONSTRAINT PROBLEMS USING BOOLEAN MODELS

The Use of Ternary Vectors for Parallel SAT-Solving

Christian Posthoff

*Department of Mathematics & Computer Science, The University of the West Indies, Trinidad & Tobago
Christian.Posthoff@sta.uwi.edu*

Bernd Steinbach

*Institute of Computer Science, Freiberg University of Mining and Technology, Freiberg, Germany
steinb@informatik.tu-freiberg.de*

Keywords: constraint problems, Boolean models, ternary vectors, intersection, bit-parallel, XBOOLE.

Abstract: The use of Boolean models for discrete constraint problems has been tried at several occasions, it was, however, not recognized as efficient (Rossi et al., 2006). The solution methods were dominated by using decision trees together with depth-first or breadth-first search and/or resolution algorithms. In this paper we will show the use of ternary vectors for the solution of SAT-problems and all the problems that can be modeled by means of SAT-equations. They are an appropriate data structure representing sets of Boolean vectors. They also allow to include problem-relevant knowledge into the problem-solving process at an early point of time. The respective set operations (mainly the intersection) can be executed in a *bit-parallel* way (64 bits at present). For larger problems the processing can be transferred to processors working fully in parallel. There is no need for any search algorithms. The approach always finds **all** solutions of the problem without consideration of special cases (i.e. no solution, one solution, all solutions). Some examples are used to illustrate the approach or have been published before (Sudoku, Queen's problems on the chessboard, node bases in graphs, graph-coloring problems).

1 INTRODUCTION

The satisfiability problem (SAT) has been very well explored. Besides of special tools for SAT solving we apply in this paper the more general tool XBOOLE (Posthoff and Steinbach, 2004), (Steinbach and Posthoff, 2009). XBOOLE allows to solve logic equations and manipulate solution sets efficiently. The efficiency of solving complex problems results particularly from the compression of sets in ternary vectors and their bit-parallel computation. Additionally it is possible to distribute the problem to be solved to several processors (Posthoff and Steinbach, 2006).

Based on the mentioned very efficient parallel algorithms for its solution, it will be shown that many combinatorial problems can be transformed into satisfiability problems and solved using these developed algorithms. The approach is constructive and very general, no research procedures are involved, and the results are always complete.

As extension to the classical SAT approach we suggest a new modeling approach called *two-phase*

SAT solver which works on a higher level and utilizes the efficient operations of XBOOLE.

2 TERNARY VECTORS AS THE MAIN DATA STRUCTURE

As a first step we introduce the data structure of a *ternary vector*. Let $\mathbf{x} = (x_1, \dots, x_n)$, $x_i \in \{0, 1, -\}$, $i = 1, \dots, n$. Then \mathbf{x} is called a **ternary vector** which can be understood as an abbreviation of a set of binary vectors. When we replace each $-$ by 0 or 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 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 there is given a conjunction C with variables x_1, \dots, x_k , then we can build a ternary vector t with the components

t_1, \dots, t_k according to the following coding:

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

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

Example: Let be given $x_1\bar{x}_2x_3\bar{x}_5 = 1$, then we have $\mathbf{t} = (101-0)$ 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 \mathbf{x} and \mathbf{y} . The intersection of these two vectors (i.e. the intersection of the respective two sets of binary vectors) will be computed according to Table 1 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 and can be omitted. A sophisticated coding of

Table 1: 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 three values 0, 1 and - allows 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 2. The first bit indicates that the variable has a value in the ternary vector, the second bit indicates the value itself.

Table 2: 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

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

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

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

$$\text{bit2}(\mathbf{x} \cap \mathbf{y}) = \text{bit2}(\mathbf{x}) \vee \text{bit2}(\mathbf{y}) \quad (4)$$

Hint: \oplus indicates the exclusive-or, $\mathbf{0}$ 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

Using these ternary vectors as the basic data structure, we are able to solve SAT-problems directly. We will use the following small example:

$$(a \vee \bar{b} \vee \bar{c})(b \vee \bar{d} \vee \bar{e})(\bar{a} \vee d \vee e)(b \vee c \vee \bar{e}) = 1 . \quad (5)$$

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

$$a \vee \bar{b} \vee \bar{c} = 1 , \quad (6)$$

$$b \vee \bar{d} \vee \bar{e} = 1 , \quad (7)$$

$$\bar{a} \vee d \vee e = 1 , \quad (8)$$

$$b \vee c \vee \bar{e} = 1 . \quad (9)$$

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

$$\begin{pmatrix} a & b & c & d & e \\ 1 & - & - & - & - \\ 0 & 0 & - & - & - \\ 0 & 1 & 0 & - & - \end{pmatrix}$$

This matrix shows all the vectors that satisfy the first equation. If $a = 1$, then the values of the other variables are not important. If $a = 0$, then \bar{b} must be equal to 1, i. e. $b = 0$. Finally, if $a = 0$ and $b = 1$, then c must be equal to 0. This construction has the additional property (advantage) that every pair of vectors of this matrix has an empty intersection, and therefore any double solutions cannot exist. \bar{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.

$$\text{solution of equation (6)} \quad \begin{pmatrix} a & b & c & d & e \\ 1 & - & - & - & - \\ 0 & 0 & - & - & - \\ 0 & 1 & 0 & - & - \end{pmatrix}$$

$$\text{solution of equation (7)} \quad \begin{pmatrix} a & b & c & d & e \\ - & 1 & - & - & - \\ - & 0 & - & 0 & - \\ - & 0 & - & 1 & 0 \end{pmatrix}$$

$$\text{solution of equation (8)} \quad \begin{pmatrix} a & b & c & d & e \\ 0 & - & - & - & - \\ 1 & - & - & 1 & - \\ 1 & - & - & 0 & 1 \end{pmatrix}$$

$$\text{solution of equation (9)} \quad \begin{pmatrix} a & b & c & d & e \\ - & 1 & - & - & - \\ - & 0 & 1 & - & - \\ - & 0 & 0 & - & 0 \end{pmatrix}$$

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:

$$\begin{pmatrix} a & b & c & d & e \\ 1 & - & - & - & - \\ 0 & 0 & - & - & - \\ 0 & 1 & 0 & - & - \end{pmatrix} \cap \begin{pmatrix} a & b & c & d & e \\ - & 1 & - & - & - \\ - & 0 & - & 0 & - \\ - & 0 & - & 1 & 0 \end{pmatrix} = \begin{pmatrix} a & b & c & d & e \\ - & 0 & - & 1 & 0 \\ - & 0 & - & 0 & - \\ 0 & 1 & 0 & - & - \\ 1 & 1 & - & - & - \end{pmatrix}, \quad (10)$$

and the final result is equal to

$$\begin{pmatrix} a & b & c & d & e \\ 0 & 0 & - & - & 0 \\ - & 0 & 1 & 0 & 1 \\ 1 & 1 & - & - & 1 \\ 0 & 1 & 0 & - & - \\ 1 & - & - & 1 & 0 \end{pmatrix}. \quad (11)$$

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.

4 SUDOKU AS AN EXAMPLE

Over the last years a Japanese game with the name **Sudoku** became very popular. It is played mostly on a board with 9×9 fields, 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

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.

also mathematical and logical properties that deserve some attention.

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

Some values already have been set, and 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 bottom-up (in the same way as a normal planar coordinate system).

We know at least two papers that are using SAT for the modeling of the game and existing SAT-solvers for the solution of the problem (Lynce and Ouaknine, 2006), (Weber, 2005). These papers used the following approach: a binary variable x_{ijk} describes the content of the field (i, j) , $1 \leq i, j, k \leq 9$:

$$x_{ijk} = \begin{cases} 1 & \text{if the value on the field } (i, j) = k \\ 0 & \text{if the value on the field } (i, j) \neq k \end{cases}. \quad (12)$$

The transformation into a SAT-problem uses several steps:

$$\begin{aligned} &x_{i1} \vee x_{i2} \vee x_{i3} \vee x_{i4} \vee x_{i5} \\ &\vee x_{i6} \vee x_{i7} \vee x_{i8} \vee x_{i9} = 1. \end{aligned} \quad (13)$$

expresses the requirement that one of the numbers $1, \dots, 9$ must be used for the field (i, j) . Such a disjunction must be written for each field 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 field $(1, 1)$ and the value 1 on this field, no other value can be on this field:

$$\begin{aligned} x_{111} &\rightarrow \bar{x}_{112}, x_{111} \rightarrow \bar{x}_{113}, \dots, \\ x_{111} &\rightarrow \bar{x}_{118}, x_{111} \rightarrow \bar{x}_{119}. \end{aligned} \quad (14)$$

The same set of clauses must be written for the other values $2, \dots, 9$ on the same field. The requirements for the first column can be expressed in the

same way:

$$\begin{aligned} x_{111} &\rightarrow \bar{x}_{121}, x_{111} \rightarrow \bar{x}_{131}, \dots, \\ x_{111} &\rightarrow \bar{x}_{181}, x_{111} \rightarrow \bar{x}_{191} . \end{aligned} \quad (15)$$

The constraints for the row are given as

$$\begin{aligned} x_{111} &\rightarrow \bar{x}_{211}, x_{111} \rightarrow \bar{x}_{311}, \dots, \\ x_{111} &\rightarrow \bar{x}_{811}, x_{111} \rightarrow \bar{x}_{911} , \end{aligned} \quad (16)$$

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

$$\begin{aligned} x_{111} &\rightarrow \bar{x}_{221}, x_{111} \rightarrow \bar{x}_{231}, \\ x_{111} &\rightarrow \bar{x}_{321}, x_{111} \rightarrow \bar{x}_{331} . \end{aligned} \quad (17)$$

Again all these clauses have to be written for all numbers from 1 to 9 and finally for all fields. By using the rule $x \rightarrow y = \bar{x} \vee y$ the whole set of implications can be transformed into disjunctions, all of them must be satisfied 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**.

We are using the same encoding (12) as the two other papers mentioned above. The constraints can be stated by one single conjunction for each number on each field:

$$\begin{aligned} K_{111} = & x_{111} \wedge \\ & \bar{x}_{112}\bar{x}_{113}\bar{x}_{114}\bar{x}_{115}\bar{x}_{116}\bar{x}_{117}\bar{x}_{118}\bar{x}_{119} \wedge \\ & \bar{x}_{121}\bar{x}_{131}\bar{x}_{141}\bar{x}_{151}\bar{x}_{161}\bar{x}_{171}\bar{x}_{181}\bar{x}_{191} \wedge \\ & \bar{x}_{211}\bar{x}_{311}\bar{x}_{411}\bar{x}_{511}\bar{x}_{611}\bar{x}_{711}\bar{x}_{811}\bar{x}_{911} \wedge \\ & \bar{x}_{221}\bar{x}_{231}\bar{x}_{321}\bar{x}_{331} . \end{aligned} \quad (18)$$

This conjunction describes completely the setting of 1 on the field (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 requirement in terms of 9 variable x_{ijk} are taken into consideration, but the conjunctions K_{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

$$\begin{aligned} & K_{111} \vee K_{112} \vee K_{113} \vee K_{114} \vee \\ & K_{115} \vee K_{116} \vee K_{117} \vee K_{118} \vee K_{119} = 1 \end{aligned} \quad (19)$$

describes that one of the 9 values must be assigned to one field (the field (11) is only an example).

2. The equation

$$\begin{aligned} & K_{111} \vee K_{121} \vee K_{131} \vee K_{141} \\ & \vee K_{151} \vee K_{161} \vee K_{171} \vee K_{181} \vee K_{191} = 1 \end{aligned} \quad (20)$$

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

3. The equation

$$\begin{aligned} & K_{111} \vee K_{211} \vee K_{311} \vee K_{411} \\ & \vee K_{511} \vee K_{611} \vee K_{711} \vee K_{811} \vee K_{911} = 1 \end{aligned} \quad (21)$$

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

4. The equation

$$\begin{aligned} & K_{111} \vee K_{121} \vee K_{131} \vee K_{211} \vee K_{221} \\ & \vee K_{231} \vee K_{311} \vee K_{321} \vee K_{331} = 1 \end{aligned} \quad (22)$$

describes that the values 1 must be assigned to one of the fields 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, one system can be selected once and for ever. 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.

Since the values which have already been set result in one vector for the given field, many of these matrices will have only one row (30 in the given example). Therefore it is advisable (however, not necessary), to intersect the single vectors first, because thereafter many intersections will be empty. This algorithm does not need any considerations for special cases. If there are no solutions, then the intersection will be empty at a given point of time, if there is a unique solution, then we will have precisely one vector in the final intersection, and more than one solution will be expressed by the respective number of vectors.

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 that indicate the solution numbers associated to the fields. The remaining 648 components of each solution vector carry the value 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 field (i, j) of column i and row j can be established.

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

The second phase mainly considers the different action possibilities and combines these possibilities by \vee . The intersections of the different possible actions are evaluated by the intersections of the respective ternary vectors.

The advantage of this new approach in comparison with the known traditional SAT-models is the simultaneous assignment of values to many variables. In 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.

5 EXPERIMENTAL RESULTS

In 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 4096 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 4096 variables and 111616 clauses could be solved within about two and a half minutes.

6 OTHER 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 size 8×8 .

The pawn interrupts the effective lines of the queens, and the diagram really shows 9 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 3 summarizes the experimental results for chessboards of several sizes.

							Q
		Q					
				Q			
	Q			P		Q	
				Q			
Q							
					Q		
			Q				

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

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.

2. The case of $n = 0$ 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 bishops can at most be placed on a chessboard without covering each other etc. These problems also have been solved on boards of size $m \times n$ for many values of m, n .
4. The same relates to graph problems, such as Hamiltonian and Eulerian paths in a graph, the minimum number of nodes with a given property or the maximum number of nodes with a given property etc.
5. As a last example we will show the solution of coloring problems. Our method can be applied to color any graph. As example we use the graph

							Q
				Q			
Q		Q		P	Q		
			Q				
	Q	P				Q	
				Q			
		Q					

Figure 4: Example of a solution of 10 queens and 2 pawns.

Table 3: Number of variables and solutions for 2 pawns and the maximal number of queens on chessboards of size $n \times n$.

n	# variables	# solutions	time in sec
3	9	0	0.00
4	16	0	0.0
5	25	0	0.0
6	36	0	0.0
7	49	4	0.20
8	64	44	0.92
9	81	280	3.31
10	100	1304	11.09
11	121	12452	97.21
12	144	105012	406.07

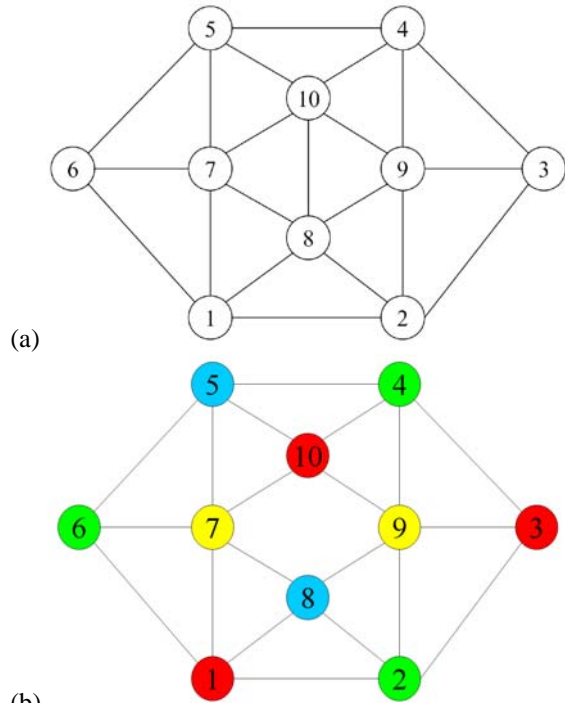


Figure 5: Birkhoff's diamond: (a) uncolored graph, (b) one solution using 4 colors.

called *Birkhoff's Diamond* shown in Figure 5 (a).

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 case of an undirected graph we get a symmetric adjacency matrix. The graph *Birkhoff's Diamond* has the following adjacency matrix.

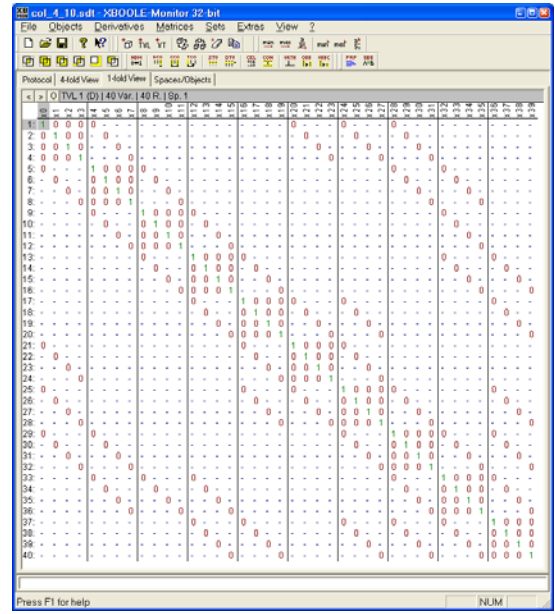


Figure 6: Generated matrix of the partial solution sets to color the graph of Birkhoff's diamond using four colors.

$$A_{BD} = \begin{pmatrix} 0100011100 \\ 1010000110 \\ 0101000010 \\ 0010100011 \\ 0001011001 \\ 1000101000 \\ 1000110101 \\ 1100001011 \\ 0111000101 \\ 0001101110 \end{pmatrix} \quad (23)$$

Using the adjacency matrix (23) the partial solution sets can be generated directly. The logic variables describe whether a certain color is assigned to the 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.

$$x_{cn} = \begin{cases} 1 & \text{if the color on the node } n = c \\ 0 & \text{if the color on the node } n \neq c \end{cases} \quad (24)$$

These partial solution sets cover the the restrictive rules. When a color is assigned to one node, it is not allowed that:

- (a) another color is assigned to the same node, and
- (b) the some color is assigned to another node connected by an edge.

Figure 6 shows the generated matrix of the partial solution sets (*mpss*) in the XBOOLE mon-

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

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

Table 5: Calculation of all solutions to color the Birkhoff’s diamond and graphs that include two or four such graphs using 4 colors.

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

itor (Posthoff and Steinbach, 2004), (Steinbach and Posthoff, 2009). Each row represents a conjunction K_{cn} covering seven, eight or nine components.

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 four colors for the graph of Figure 5, we must solve the equation:

$$\bigwedge_{i=1}^{10} (K_{1i} \vee K_{2i} \vee K_{3i} \vee K_{4i}) = 1 \quad . \quad (25)$$

The time to solve this equation using the operations UNI and ISC of XBOOLE (Posthoff and Steinbach, 2004) 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 4 summarizes these results.

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. Table 5 summarizes these results.

7 SUMMARY

There are several results presented in this paper.

1. Many finite discrete constraint-related problems can be modeled as a SAT-problem. It has been shown that it is not necessary to write down the huge number of clauses of the conjunctive forms which must be solved by a SAT-solver. Based on the explored properties of the problem, it is possible to generate partial solution sets of the restrictive properties of the problem.
2. A new implicit two-phase SAT-solver has been used. In the first phase this SAT-solver creates partial solution sets which are used in the second phase to calculate the solution without any further decisions.
3. The matrix of the partial solution sets describes general constraints of the problem without any consideration of clauses.
4. The use of the partial solution sets in the second phase of the SAT-solver allows to solve the SAT-problems very fast. The partial solution sets help to restrict significantly the enormous search space. The remaining clauses of the problem are replaced by unions of partial solution sets which speed up the solution procedure.
5. 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 ternary vectors.

REFERENCES

- Lynce, I. and Ouaknine, J. (2006). Sudoku as a sat problem. In *Proceedings of the 9 th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale*. Springer.
- Posthoff, C. and Steinbach, B. (2004). *Logic Functions and Equations - Binary Models for Computer Science*. Springer, Dordrecht, The Netherlands.
- Posthoff, C. and Steinbach, B. (2006). A multi-processor approach to sat-problems. In *Proceedings of the 7th International Workshop on Boolean Problems*, pages 49–62, Freiberg University of Mining and Technology, Freiberg, Germany.
- Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming*. ELSEVIER.
- Steinbach, B. and Posthoff, C. (2009). *Logic Functions and Equations - Examples and Exercises*. Springer Science + Business Media B.V.
- Weber, T. (2005). A SAT-based Sudoku solver. In Sutcliffe, G. and Voronkov, A., editors, *LPAR-12, The 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, Short Paper Proceedings*, pages 11–15.