FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 20, no. 3, December 2007, 395-414

Set-Based SAT-Solving

Bernd Steinbach and Christian Posthoff

Abstract: The 3-SAT problem is one of the most important and interesting NP-complete problems with many applications in different areas. In several previous papers we showed the use of ternary vectors and set-theoretic considerations as well as binary codings and bit-parallel vector operations in order to solve this problem. Lists of orthogonal ternary vectors have been the main data structure, the intersection of ternary vectors (representing sets of binary solution candidates) was the most important set-theoretic operation. The parallelism of the solution process has been established on the register level, i.e. related to the existing hardware, by using a binary coding of the ternary vectors, and it was also possible to transfer the solution process to a hierarchy of processors working in parallel. This paper will show further improvements of the existing algorithms which are easy to understand and result in very efficient algorithms and implementations. Some examples will be presented that will show the recent results.

Keywords: SAT-problem, ternary vector, intersection, difference.

1 Introduction

Boolean problems combine practical and theoretical aspects in a special way. Konrad Zuse [7] revealed the practical importance that two different values of a binary variable can be safely distinguished in a physical system. This understanding led to the breakthrough of modern computer science. The first fully functional electromechanical digital computer in the world (the Z3) was completed by Zuse in 1941. Recently, the binary representation and processing of information is used in more and more areas of our life. Computers can process 64-bit numbers directly in a

Manuscript received August 4, 2007.

B. Steinbach is with Institute of Computer Science, University of Mining and Technology Freiberg, Bernhard-von-Cotta-Str. 2, D-09596 Freiberg, Germany (e-mail: steinb@informatik.tu-freiberg.de). C. Posthoff is with the Department of Mathematics & Computer Science, University of The West Indies, Trinidad & Tobago, (e-mail: Christian.Posthoff@sta.uwi.edu).

single step. Programs, data, music and movies are stored on CDs or DVDs using binary values. Phone calls, faxes, files and pictures are transmitted as sequences of binary values using ISDN or the Internet, and the traditional analogous technologies of radio or TV will be substituted by digital systems having a much higher quality and requiring less resources.

In addition to the theories of Boolean Algebras and the Boolean Differential Calculus [5], certain Boolean problems appear as a key in complexity theory [6]. The *Boolean satisfiability problem (SAT)* was the first known *NP-complete problem*. A decision problem is in the complexity class NP if a non-deterministic Turing machine can solve it in polynomial time. A decision problem is NP-complete if it is in NP and every problem in NP can be reduced [3] to it by a polynomial-time reduction. Stephen Cook proved that the Boolean satisfiability problem is NP-complete [1]. This theorem was independently proven by Leonid Levin [4] at about the same time, therefore it is called Cook-Levin theorem. NP-complete problems are the most interesting problems in NP. In the context of the practical importance mentioned above we suggest in this paper an approach to solve large SAT-problems as efficient as possible.

2 Preliminaries

In order to make this note independently readable, we summarize the concepts that have been used previously. 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* 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.

A Boolean expression is said to be *satisfiable* if binary values 0 or 1 can be assigned to its variables in a way that makes the expression equal to 1 (true). Both a variable and its negation are called *literals*. A disjunction of literals is called *clause*. A Boolean expression is given in *conjunctive form* if it only consists of clauses connected by AND-operators. The check for satisfiability (SAT) is NP-complete if the Boolean expression is given in conjunctive form with three or more variables in its clauses. In the special case of the 3-SAT problem each clause includes three literals. This problem is particularly important because (by increasing the number of variables) each SAT-problem can be transformed into a 3-SAT-problem.

If we consider now one clause of a 3-SAT problem, such as $C = x_2 \vee \overline{x}_3 \vee x_4$, then we find all solutions of $C = 1$ in the following way:

- If $x_2 = 1$, then $C = 1$ independent on the other variables. Hence, $(-1--)$ describes a set of solutions.
- If $x_2 = 0$, then $\overline{x}_3 = 1$, i.e. $x_3 = 0$ gives more solutions that are not covered by the previous case and described by $(-00-)$.
- Finally, if $x_2 = 0$ and $x_3 = 1$, then x_4 must be equal to 1. Thus, (-011) is another set of solutions.

The three solution sets do not have common elements, the intersections of always two of these sets is empty, i.e. we have *disjoint* solution sets. This property can easily be seen from the representing ternary vectors: in at least one of the components of the two representing vectors we find the combination of values 0 and 1.

If we consider now two clauses $C_1C_2 = 1$, then we build the solution sets of $C_1 = 1$ and $C_2 = 1$ and find the solution of $C_1C_2 = 1$ as intersection of the respective solution sets.

The intersection will be computed according to Table 1 which has to be applied in each component. $\emptyset$ indicates that the intersection is empty and can be omitted.

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	–

A sophisticated coding of 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.

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 empty intersection can be determined by

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

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

$$bit1(\mathbf{x} \cap \mathbf{y}) = bit1(\mathbf{x}) \vee bit1(\mathbf{y}), \quad bit2(\mathbf{x} \cap \mathbf{y}) = bit2(\mathbf{x}) \vee bit2(\mathbf{y}).$$

Hence, by using some very fast and very simple bit vector operations, we can find the solution sets of any SAT-equations, and especially of 3-SAT-equations. In [2] this could be done for a maximum of up to 280 variables in about 10 minutes.

3 Analysis

One problem, however, occurred for some SAT-problems related to the representation of intermediate results (sets of solution candidates). The number of solution candidates can and will get smaller and smaller, particularly when we assume a small set of final solutions. The number of intermediate ternary vectors, however, can be very large and require large amounts of memory space and computing time to process these sets of ternary vectors.

In the following we will see that the overlap of the variables, i.e. the number of common variables, in different disjunctions will be important for the growth of intermediate results. At first we want to see the frequency of these overlaps.

Let us consider n variables, then we will find $\left(\begin{array}{c} n \\ 3 \end{array} \right)$ different combinations of variables:

$$\left(\begin{array}{c} n \\ 3 \end{array} \right) = \frac{n(n-1)(n-2)}{6} \approx n^3.$$

In order to determine the chances for an overlap, we consider

$$\Omega(n) = \left(\begin{array}{c} n \\ 3 \end{array} \right) \left(\begin{array}{c} n \\ 3 \end{array} \right) = \frac{n(n-1)(n-2)n(n-1)(n-2)}{6 \cdot 6} \approx n^6$$

possible pairs of disjunctions.

A. No overlap

In order to get disjoint sets of variables in the two disjunctions, we must have $n \geq 6$, and then we find

$$\Omega_0(n) = \left(\begin{array}{c} n \\ 3 \end{array} \right) \left(\begin{array}{c} n-3 \\ 3 \end{array} \right) = \frac{n(n-1)(n-2)(n-3)(n-4)(n-5)}{6 \cdot 6} \approx n^6$$

different combinations. In order to generalize this analysis we introduce

$$\left(\begin{array}{c} n \\ 0 \end{array} \right) = 1$$

in this formula and get for no overlap

$$\Omega_0(n) = \binom{n}{0}\binom{n}{3}\binom{n-3}{3}.$$

B. One overlap

We have n single positions for the overlap, the remaining $n-1$ positions are available for 2 variables, hence, $n\binom{n-1}{2}$ first disjunctions. The second disjunction must use the same position for the overlap, 2 more positions are forbidden, hence, $\binom{n-3}{2}$ possible second disjunctions. Altogether we have

$$\Omega_1(n) = n\binom{n-1}{2}\binom{n-3}{2} = \frac{n(n-1)(n-2)(n-3)(n-4)}{4} \approx n^5$$

possible pairs of disjunctions overlapping in one variable. In order to continue the generalization we use

$$\binom{n}{1} = n$$

in this formula and get for one overlap

$$\Omega_1(n) = \binom{n}{1}\binom{n-1}{2}\binom{n-3}{2}.$$

C. Two overlaps

We have $\binom{n}{2}$ possible combinations for the overlapping positions, the third variable can use $(n-2)$ positions. In the second disjunction the overlapping positions are the same, the third variable can use $(n-3)$ positions. Therefore we get

$$\Omega_2(n) = \binom{n}{2}(n-2)(n-3) = \frac{n(n-1)(n-2)(n-3)}{2} \approx n^4$$

possible pairs overlapping in two variables. Using

$$\binom{n-2}{1} = n-2 \quad \text{and} \quad \binom{n-3}{1} = n-3$$

we get for two overlaps

$$\Omega_2(n) = \binom{n}{2}\binom{n-2}{1}\binom{n-3}{1}.$$

Table 3. Distribution of Overlapping

	n=10	percentage	n=20	percentage
$\Omega(n)$	14,400	100.00	1,299,600	100.00
$\Omega_0(n)$	4,200	29.17	775,200	59.65
$\Omega_1(n)$	7,560	52.50	465,120	35.79
$\Omega_2(n)$	2,520	17.50	58,140	4.47
$\Omega_3(n)$	120	0.83	1,140	0.09

D. Three overlaps

Each combination of three variables can overlap only itself, hence we have

$$\Omega_3(n) = \left(\begin{array}{c} n \\ 3 \end{array} \right) = \frac{n(n-1)(n-2)}{6} \approx n^3$$

pairs of disjunctions with three overlapping variables. In order to find a general solution we use

$$\left(\begin{array}{c} n-3 \\ 0 \end{array} \right) = 1$$

twice and get for three overlaps

$$\Omega_3(n) = \left(\begin{array}{c} n \\ 3 \end{array} \right) \left(\begin{array}{c} n-3 \\ 0 \end{array} \right) \left(\begin{array}{c} n-3 \\ 0 \end{array} \right).$$

As general distribution of possible combinations of clauses overlapping in m variables, $0 \leq m \leq 3$, for solving a 3-SAT-problem we get

$$\Omega_m(n) = \left(\begin{array}{c} n \\ m \end{array} \right) \left(\begin{array}{c} n-m \\ 3-m \end{array} \right) \left(\begin{array}{c} n-3 \\ 3-m \end{array} \right). \tag{1}$$

For smaller n the overlap is quite remarkable. For larger n we can see in Table 3 that the overlap more and more will be a rare case. This observation takes into account the comparison of exactly two clauses. For a large number of clauses and a large number of variables n, there is still a significant number of overlaps.

Now we will explore the effect of the overlap.

A. Two disjunctions, no overlap: nine orthogonal solution vectors.

$$f(a,b,c,d,e,f) = (a \vee b \vee c)(d \vee e \vee f)$$

$$\left(\begin{array}{cccccc} a & b & c & d & e & f \\ \hline 1 & - & - & - & - & - \\ 0 & 1 & - & - & - & - \\ 0 & 0 & 1 & - & - & - \end{array}\right) \cap \left(\begin{array}{cccccc} a & b & c & d & e & f \\ \hline - & - & - & 1 & - & - \\ - & - & - & 0 & 1 & - \\ - & - & - & 0 & 0 & 1 \end{array}\right) \Rightarrow$$

$$\left(\begin{array}{cccccc} a & b & c & d & e & f \\ \hline 1 & - & - & 1 & - & - \\ 1 & - & - & 0 & 1 & - \\ 1 & - & - & 0 & 0 & 1 \\ 0 & 1 & - & 1 & - & - \\ 0 & 1 & - & 0 & 1 & - \\ 0 & 1 & - & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & - & - \\ 0 & 0 & 1 & 0 & 1 & - \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right)$$

This is the crucial situation for the large increase of intermediate lists of vectors.

B.1 Two disjunctions, one overlap, the overlap variable either non-negated in both disjunctions or negated in both disjunctions: five orthogonal solution vectors.

$$f(a,b,c,d,e) = (a \vee b \vee c)(c \vee d \vee e)$$

$$\left(\begin{array}{ccccc} a & b & c & d & e \\ \hline - & - & 1 & - & - \\ 1 & - & 0 & - & - \\ 0 & 1 & 0 & - & - \end{array}\right) \cap \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline - & - & 1 & - & - \\ - & - & 0 & 1 & - \\ - & - & 0 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline - & - & 1 & - & - \\ 1 & - & 0 & 1 & - \\ 1 & - & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & - \\ 0 & 1 & 0 & 0 & 1 \end{array}\right)$$

$$f(a,b,c,d,e) = (a \vee b \vee \overline{c})(\overline{c} \vee d \vee e)$$

This case can be reduced to the previous case using the substitution $\overline{c} = x$ which results in

$$f(a,b,x,d,e) = (a \vee b \vee x)(x \vee d \vee e),$$

with exactly the same results. For the minimal set of solution vectors the overlap variable has to be used first, the other variables can follow in any order.

B.2 Two disjunctions, one overlap, the overlap variable once negated and once non-negated: four orthogonal solution vectors.

$$f(a,b,c,d,e) = (a \vee b \vee c)(\overline{c} \vee d \vee e)$$

$$\left(\begin{array}{ccccc} a & b & c & d & e \\ \hline - & - & 1 & - & - \\ 1 & - & 0 & - & - \\ 0 & 1 & 0 & - & - \end{array}\right) \bigcap \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline - & - & 0 & - & - \\ - & - & 1 & 1 & - \\ - & - & 1 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline - & - & 1 & 1 & - \\ - & - & 1 & 0 & 1 \\ 1 & - & 0 & - & - \\ 0 & 1 & 0 & - & - \end{array}\right)$$

For the orthogonal representation the overlap variable has to be used first, the other variables can follow in any order.

C.1 Two disjunctions, two overlaps, the overlap variables both non-negated (or both negated): three orthogonal solution vectors.

$$f(a,b,c,d) = (a \vee b \vee c)(b \vee c \vee d)$$

$$\left(\begin{array}{cccc} a & b & c & d \\ \hline - & 1 & - & - \\ - & 0 & 1 & - \\ 1 & 0 & 0 & - \end{array}\right) \bigcap \left(\begin{array}{cccc} a & b & c & d \\ \hline - & 1 & - & - \\ - & 0 & 1 & - \\ - & 0 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{cccc} a & b & c & d \\ \hline - & 1 & - & - \\ - & 0 & 1 & - \\ 1 & 0 & 0 & 1 \end{array}\right)$$

For the orthogonal representation the overlap variables have to be used first. If the two overlap variables are both negated, an appropriate substitution, such as $\overline{b} = x$, $\overline{c} = y$, has to be used.

C.2 Two disjunctions, two overlaps, one overlap variable negated and non-negated, the second variable negated or non-negated twice: four orthogonal solution vectors.

$$f(a,b,c,d) = (a \vee b \vee c)(\overline{b} \vee c \vee d)$$

$$\left(\begin{array}{cccc} a & b & c & d \\ \hline - & 1 & - & - \\ - & 0 & 1 & - \\ 1 & 0 & 0 & - \end{array}\right) \bigcap \left(\begin{array}{cccc} a & b & c & d \\ \hline - & 0 & - & - \\ - & 1 & 1 & - \\ - & 1 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{cccc} a & b & c & d \\ \hline - & 1 & 1 & - \\ - & 1 & 0 & 1 \\ - & 0 & 1 & - \\ 1 & 0 & 0 & - \end{array}\right)$$

For the orthogonal representation the overlap variables have to be used first. If the second overlap variable is negated in both disjunctions, an appropriate substitution, such as $\overline{c} = x$, has to be used.

D.1 Two disjunctions, three overlaps, all three variables non-negated or all three variables negated: three orthogonal solution vectors.

In this case the two disjunctions are equal to each other, the second disjunction can be deleted, the first disjunction will have three orthogonal solution vectors.

D.2 Two disjunctions, three overlaps, one variable negated and non-negated: four orthogonal solution vectors.

$$f(a,b,c) = (a \vee b \vee c)(\overline{a} \vee b \vee c)$$

$$\left(\begin{array}{ccc} a & b & c \\ \hline 1 & - & - \\ 0 & 1 & - \\ 0 & 0 & 1 \end{array}\right) \bigcap \left(\begin{array}{ccc} a & b & c \\ \hline 0 & - & - \\ 1 & 1 & - \\ 1 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc} a & b & c \\ \hline 1 & 1 & - \\ 1 & 0 & 1 \\ 0 & 1 & - \\ 0 & 0 & 1 \end{array}\right)$$

Here the result is very simple, the second and the third vector of the two initial matrices will be taken in combination with the first of the other initial matrix.

D.3 Two disjunctions, three overlaps, two variables negated and non-negated: four orthogonal solution vectors.

$$f(a,b,c) = (a \vee b \vee c)(\overline{a} \vee \overline{b} \vee c)$$

$$\left(\begin{array}{ccc} a & b & c \\ \hline 1 & - & - \\ 0 & 1 & - \\ 0 & 0 & 1 \end{array}\right) \bigcap \left(\begin{array}{ccc} a & b & c \\ \hline 0 & - & - \\ 1 & 0 & - \\ 1 & 1 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc} a & b & c \\ \hline 1 & 0 & - \\ 1 & 1 & 1 \\ 0 & 1 & - \\ 0 & 0 & 1 \end{array}\right)$$

Here we get the same number of solution vectors as before. The solution itself has changed slightly.

D.4 Two disjunctions, three overlaps, three variables negated and non-negated: four orthogonal solution vectors.

$$f(a,b,c) = (a \vee b \vee c)(\overline{a} \vee \overline{b} \vee \overline{c})$$

$$\left(\begin{array}{ccc} a & b & c \\ \hline 1 & - & - \\ 0 & 1 & - \\ 0 & 0 & 1 \end{array}\right) \bigcap \left(\begin{array}{ccc} a & b & c \\ \hline 0 & - & - \\ 1 & 0 & - \\ 1 & 1 & 0 \end{array}\right) \Rightarrow \left(\begin{array}{ccc} a & b & c \\ \hline 1 & 0 & - \\ 1 & 1 & 0 \\ 0 & 1 & - \\ 0 & 0 & 1 \end{array}\right)$$

Here we get the same number of solution vectors as before. The solution itself has again changed slightly.

4 Implementation Aspects

4.1 Restriction-based approach

In the set-based approach for solving the 3-SAT problem as introduced above partial solution sets for each clause are generated directly using an iterative or recursive algorithm. Intersections of these partial solution sets produce the wanted solution.

Alternatively an efficient restriction-based approach can be used. The 3-SAT problem $f(\mathbf{x}) = \mathbf{1}$ is given in a conjunctive form which can be expressed by a TVL where the form is indicated by the letter 'C'. As example the equation

$$f(a,b,c,d,e) = (a \vee b \vee \overline{c}) \cdot (a \vee \overline{b} \vee c) \cdot (\overline{a} \vee \overline{b} \vee e) \cdot (\overline{a} \vee b \vee d) \cdot (c \vee d \vee e) = 1$$

of a 3-SAT problem is depicted by

$$C(f) = \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 1 & 0 & - & - \\ 1 & 0 & 1 & - & - \\ 0 & 0 & - & - & 1 \\ 0 & 1 & - & 1 & - \\ - & - & 1 & 1 & 1 \end{array} \right) = 1.$$

Since these ternary vectors express disjunctions, there is no direct mapping to sets. In order to overcome this disadvantage, the negation according to the rule of de Morgan can be applied. In case of the above example we get

$$\overline{f(a,b,c,d,e)} = \overline{a}\,\overline{b}\,c \vee \overline{a}\,b\,\overline{c} \vee a\,b\,\overline{e} \vee a\,\overline{b}\,\overline{d} \vee \overline{c}\,\overline{d}\,\overline{e} = 0.$$

The execution of the negation according to de Morgan (NDM [5]) with a TVL is very simple and fast. The values 0 and 1 will be exchanged, and the form predicate changes from 'C' to 'D' ('D' with the meaning of a disjunctive form):

$$D(f) = \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline 0 & 0 & 1 & - & - \\ 0 & 1 & 0 & - & - \\ 1 & 1 & - & - & 0 \\ 1 & 0 & - & 0 & - \\ - & - & 0 & 0 & 0 \end{array} \right) = 0.$$

In a TVL in disjunctive form each ternary vector can be interpreted in two ways:

1. the elements describe the literals of the associated conjunction,
2. the whole vector describes a set of binary vectors.

In case of an homogeneous restrictive equation $D(f) = 0$ each ternary vector describes a restriction set: all binary vectors expressed by the set of a ternary vector do not belong to the solution set of the original equation. The values $a = 0, b = 0, c = 1$, for instance, result in $a \vee b \vee \overline{c} = 0$ which is not a solution of the original equation, or in $\overline{a}\,\overline{b}\,c = 1$ which is not a solution of $D(f) = 0$. A complement operation transforms such a partial restriction set into a required partial solution set. There are two algorithms that achieve this complement operation and the intersection of the partial solution set with the global solution. These algorithms will be explored in the next two subsections.

4.2 ISC-based algorithm

The ISC-based algorithm realizes basically the method described in section 3 and uses the restriction-based approach. After a single NDM-operation of the TVL given in conjunctive form we use the partial restriction set (*prs*) to create in a loop partial solution sets (*pss*) by means of a complement operation (CPL [5]) which then can immediately be used to calculate the intersection (ISC [5]) with the previous intermediate solution S_{i-i}:

$$S_i = S_{i-1} \cap pss_i = S_{i-1} \cap \overline{prs}_i. \tag{2}$$

Thus, the core of the ISC-based algorithm is

$$S[i] = ISC(S[i-1], CPL(prs[i])) \tag{3}$$

where $S_0 = 1$, represented by a single ternary vector with dashes only.

An advantage of the orthogonal ternary representation of a partial solution set is the possibility that eight partial binary solution vectors are expressed by three disjoint ternary vectors. A disadvantage of this representation is the asymmetry of the columns. This asymmetry is observable by different numbers of dashes, precisely 0, 1, and 2, in the columns of the variables given in the clause. Due to the results of the analysis in Section 3 a controlled complement operation is required which creates zero dashes in that column of the partial solution set fitting to the column of the intermediate solution matrix having the smallest number of dashes.

A further disadvantage of a partial solution set is the a priori segmentation of the set into three subsets. The disadvantageous effects of this segmentation are:

1. The time for the combination of these three subsets with each vector of the intermediate solution is three times higher than the processing of a single set.

2. There is an unnecessary segmentation of the intermediate solution matrix which requires more time and space in the following calculation steps.

The second effect can be explained using the example that has been introduced in subsection 4.1. The intermediate solution that takes into account the first four of five clauses is equal to

$$S_4 = \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 0 & - & 1 & - \\ 1 & 1 & - & - & 1 \\ 0 & 1 & 1 & - & - \\ 0 & 0 & 0 & - & - \end{array}\right),$$

and the remaining fifth clause is $(c \vee d \vee e)$.

Based on (2) and (3) we get

$$S_5 = S_4 \cap \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline - & - & 1 & - & - \\ - & - & 0 & 1 & - \\ - & - & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 0 & 1 & 1 & - \\ 1 & 0 & 0 & 1 & - \\ 1 & 1 & 1 & - & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & - & - \\ 0 & 0 & 0 & 1 & - \\ 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

It can be seen that the first row of S_4 builds solution vectors with the first and second row of the partial solution set. The created two solution sets may be expressed by a single ternary vector. The second vector of S_4 builds even three solution vectors with all three rows of the partial solution set. These three solution vectors may be expressed by a single ternary vector too. A significant reduction from 8 to 5 solution vectors is possible.

It should be mentioned that in this example each one of the six possible permutations of the representation of the partial solution set leads to the same number of solution vectors.

4.3 DIF-based algorithm

Another even more fundamental change of the solution philosophy arises when we merge the complement and intersection operation of (2) and (3) into difference operations (DIF [5]). Instead of using the intersection with the partial solution set

of three vectors we exclude the partial restriction set from the intermediate solution matrix.

$$S_i = S_{i-1} \setminus prs_i. \tag{4}$$

Thus, the core of the DIF-based algorithm is

$$S[i] = DIF(S[i-1], prs[i]) \tag{5}$$

where $S_0 = 1$ is represented again by a single ternary vector that includes only dashes. The exclusion of the non-solution vectors by means of the DIF-operation is quite easy, the vectors of the first matrix will be orthogonalized with regard to the vector to be eliminated and the common vectors are thrown away.

The disadvantages of the ISC-based algorithm change into advantages of the DIF-based algorithm. Instead of three vectors in a partial solution set there is a single vector in a partial restriction set (*prs*). This reduces the time for comparison for the set by a factor of three. Of cause, there is no asymmetry in the representation of *prs* so that no decision about the order of the columns is necessary. Finally, it is especially important that unnecessary segmentations of solution sets in the intermediate solution matrix are omitted.

This very important effect can be illustrated by solving the same task as in the ISC-based algorithm. The partial restriction set for the clause $(c \vee d \vee e)$ is $prs = (--000)$, because if each of the variables is equal to 0, then the clause is equal to 0 - no solution exists in this case. Based on (4) and (5), we get

$$S_5 = \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 0 & - & 1 & - \\ 1 & 1 & - & - & 1 \\ 0 & 1 & 1 & - & - \\ 0 & 0 & 0 & - & - \end{array}\right) \setminus (--000) = \left(\begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 0 & - & 1 & - \\ 1 & 1 & - & - & 1 \\ 0 & 1 & 1 & - & - \\ 0 & 0 & 0 & 1 & - \\ 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

The DIF-based algorithm creates directly the minimal solution. In the example the partial restriction set is orthogonal to the first three vectors of the intermediate solution matrix - these vectors remain unchanged. Only the last vector must change, it will be replaced by the two vectors $(0001-)$ and (00001). The vector (00000) has been excluded. The representation of the matrix for $S_5 = S_4 \setminus (--000)$ only needs 5 lines instead of 8. The efficiency of this 'tiny' step will be seen when we look at the experimental results.

4.4 Order of the clauses

In addition to the restriction-based approach it helps to downsize the intermediate solution matrix when the clauses are used in a well ordered way. The analysis showed that the overlap of two disjunctions in at least one variable reduces the number of required ternary vectors from nine to three, four or five. This property can now be used to make the intermediate matrices as small as possible. A very efficient strategy can be designed as follows:

1. Determine the frequency of each variable in the different clauses.
2. Sort the set of clauses according to these frequencies.

We call the implemented algorithm based on this strategy `sort0`. The application of `sort0` for an example of 50 variables and 218 clauses, for instance, showed the following frequencies for $x_1, \ldots, x_{50}$:

$$x_{14} : 22,\ x_{49} : 20,\ x_3 : 18,\ x_{22} : 18,\ x_{35} : 18,\ \ x_7 : 17, \ldots$$

Hence, the clauses have been sorted according to these frequencies: first all clauses with x_{14} followed by all clauses containing x_{49} etc. Due to the overlapping all rows have been sorted after 42 of the 50 variables have been considered. The experimental results in the next section will show an enormous increase of the efficiency.

Two clauses that overlap in one variable include 4 other variables into the intermediate matrix. If all variables of a clause are covered by these additional variables, it restricts the remaining search space. More generally, it is an advantage when a clause is included into the ordering if all of its variables are covered by the variables of the ordered matrix created so far. Hence, a more efficient strategy can be designed as follows:

1. Determine the frequency of each variable in the different clauses.
2. Sort the set of clauses according to these frequencies and include clauses independent on these frequencies, if they are covered completely by the ordered matrix created so far.

We call the implemented algorithm based on this extended strategy `sort1`. Table 4 shows that the application of `sort1` to the same example of 50 variables and 218 clauses uses much more clauses for the same restricted number of variables. Due to the overlapping all rows have been sorted by `sort1` after six of the 50 variables have been considered.

Table 4. Number of variables and clauses in the sorted matrix after consideration of the selected variable

variable	x_{14}	x_{49}	x_3	x_{32}	x_{35}	x_7
covered variables	27	37	44	49	50	50
clauses taken by `sort0`	22	42	59	71	86	97
clauses taken by `sort1`	26	81	136	180	213	218

5 Experimental Results

The experiments have been very interesting and successful. In order to be comparable, two examples given in [8] that allow to demonstrate the relationships have been chosen at random. All experiments were executed on a Pentium PC of 3 GHz. 15 msec were the smallest time interval that has been used for the measurements. We made 2 GB memory available for the storing of clauses and the computation of the solution.

The Example uf20-91. This example has 20 variables and 91 clauses.

Intersection
Maximum number of intermediate clauses: 4391.
Maximum at clause: i=22.
Computing time: 31 msec.
Difference
Maximum number of intermediate clauses: 2245.
Maximum at clause: i=22.
Computing time: 15 msec.

Figure 1 shows the size of the intermediate TVLs after the computation of each clause for both algorithms using a linear scale.

The Example uf50-218 This example has 50 variables and 218 clauses.

Intersection
Maximum number of intermediate clauses: 54,860,864.
Memory overflow at clause $i = 27$.
Computing time until overflow: 40,781 msec.

Difference
Maximum number of intermediate clauses: 87,418,986.

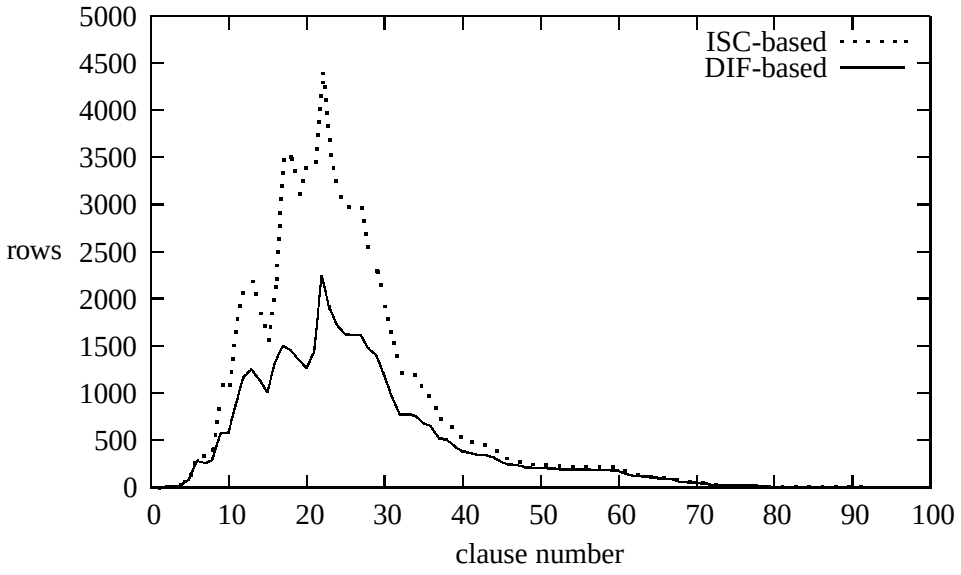

Fig. 1. Comparison of the number of ternary vectors in the intermediate TVL using the ISC- and the DIF-based algorithm for the solution of the SAT benchmark uf20-01 that depends on 20 variables and 91 clauses, on a linear scale of the rows.

Maximum at clause: i=57.
Computing time: 92,312 msec.

The comparison between the two examples above shows that that the benefit of the DIF-based algorithm grows with the increased number of variables. The ISC-based algorithm needs for the example with 20 variables approximately 2 times the peak memory of the DIF-based algorithm. A memory overflow occurs in case of the 50 variables example for the ISC-based algorithm at clause 27 where 54,860,864 intermediate solution vectors have been created. The DIF-based algorithm represents the same intermediate solution by 2,807,324 vectors after 921 msec. Hence, the ISC-based algorithm needs for the example of 50 variables before the memory overflow approximately already 20 times the memory of the DIF-based algorithm. Figure 2 shows the size of the intermediate TVLs after the computation of each clause for both algorithms using a logarithmic scale of rows.

Now we apply the algorithm `sort0` in order to find the solution faster using less memory.

Intersection and sorting by algorithm `sort0`
Maximum number of intermediate clauses: 68,644,688.

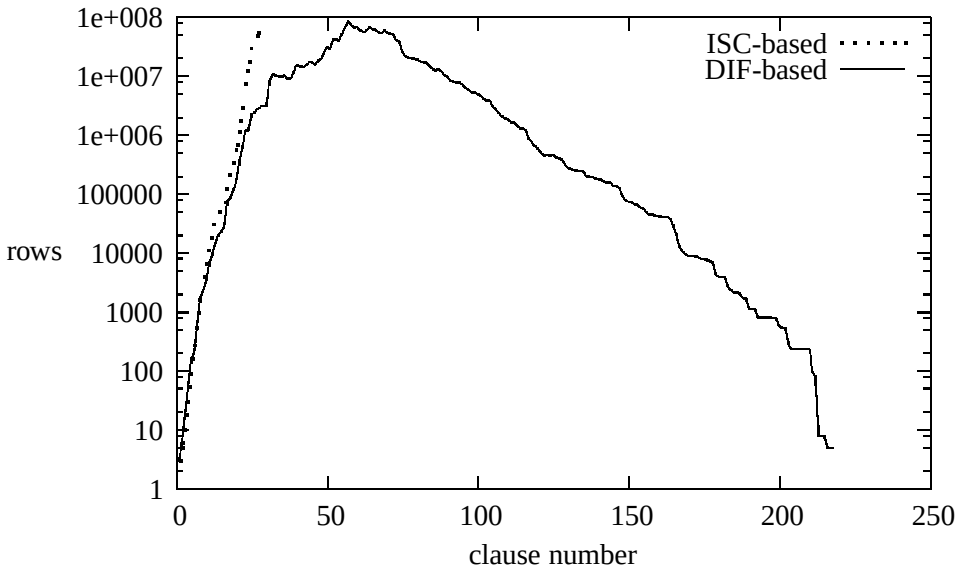

Fig. 2. Comparison of the number of ternary vectors in the intermediate TVL using the ISC-based and the DIF-based algorithm for the solution of the SAT benchmark uf50-01 that depends on 50 variables and 218 clauses, on a logarithmic scale of the rows without sorting.

Memory overflow at clause $i = 79$.
Computing time until overflow: 121,750 msec.

Difference and sorting by algorithm `sort0`
Maximum number of intermediate clauses: 658,418.
Maximum at clause: i=83.
Computing time: 968 msec.

The comparison between the examples without sorting and with sorting by `sort0` shows a gigantic improvement. The required memory for the DIF-based algorithm is reduced by applying the algorithm `sort0` by more than two orders of magnitude. A memory overflow occurs for the ISC-based algorithm with sorting by `sort0` at clause 79 where 68,644,688 intermediated solution vectors have been created. The DIF-based algorithm represents the same intermediate solution by 631,078 vectors after 390 msec. Hence, using `sort0` the ISC-based algorithm needs for the example of 50 variables before the memory overflow already more than 100 times the memory of the DIF-based algorithm.

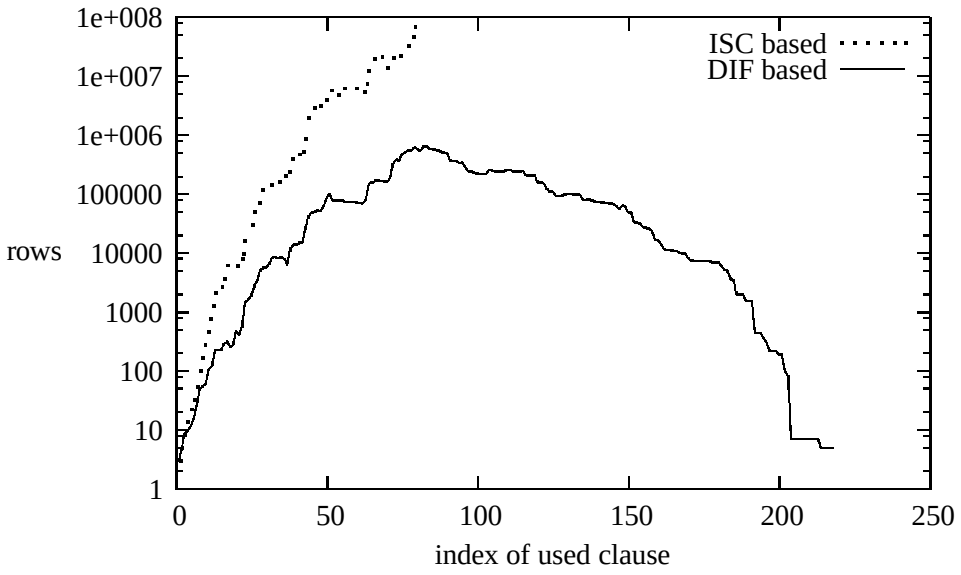

Fig. 3. Comparison of the number of ternary vectors in the intermediate TVL using the ISC-based and the DIF-based algorithm for the solution of the SAT benchmark uf50-01 that depends on 50 variables and 218 clauses, on a logarithmic scale of the rows with sorting by `sort0`.

Figure 3 shows the size of the intermediate TVLs after the computation of each clause for both algorithms applying `sort0` using a logarithmic scale of rows.

Finally we apply the algorithm `sort1` in order to evaluate further improvements with regard of time and space.

Intersection and sorting by algorithm `sort1`
Maximum number of intermediate clauses: 2,184,865.
Maximum at clause: i=90.
Computing time: 10,531 msec.

Difference and sorting by algorithm `sort1`
Maximum number of intermediate clauses: 81,741.
Maximum at clause: i=90.
Computing time: 250 msec.

The comparison between the examples with sorting by `sort0` and by `sort1` shows again a significant improvement. The ISC-based algorithm solved the ex-

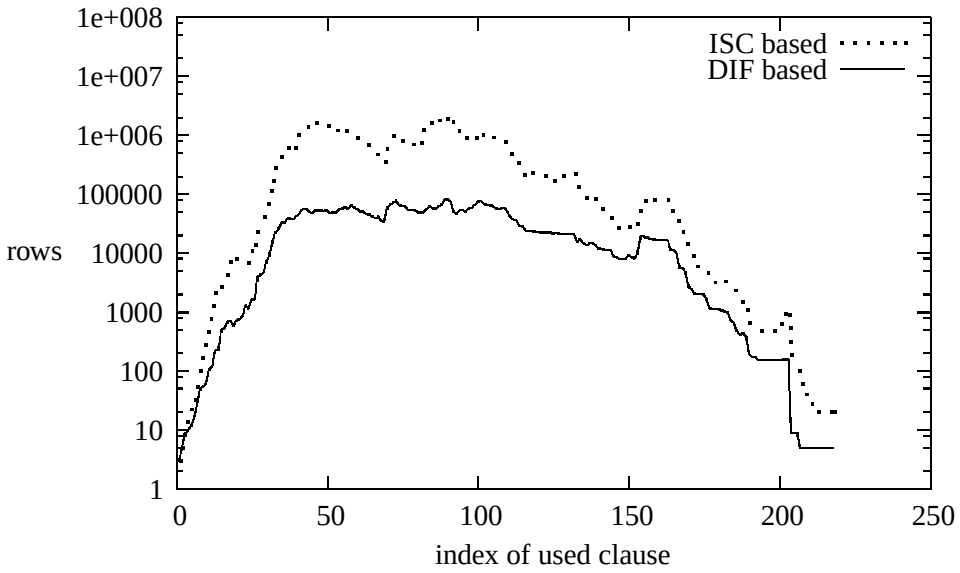

Fig. 4. Comparison of the number of ternary vectors in the intermediate TVL using the ISC-based and the DIF-based algorithm for the solution of the SAT benchmark uf50-01 that depends on 50 variables and 218 clauses, on a logarithmic scale of the rows with sorting by `sort1`.

ample of 50 variables and 218 clauses in about 10 seconds after sorting by `sort1` in contrast to memory overflows after 2 minutes and 79 clauses in case of `sort0` or after 40 seconds and 27 clauses without sorting. The required memory for the DIF-based algorithm is reduced by applying the algorithm `sort1` in comparison to the algorithm `sort0` by about one order of magnitude.

Figure 4 shows the size of the intermediate TVLs after the computation of each clause for both algorithms applying `sort1` using a logarithmic scale of rows.

6 Conclusions

Two sources were consequently used in this paper to improve the power of set-based SAT solvers. At first, a systematic analysis of the elementary task to be solved by a set-based SAT solver revealed the potential of ordering and requirements for the representation of the partial solution sets. Secondly, the analysis of the restriction-based approach indicated the advantages of the DIF-based algorithm in opposition to the classical ISC-based algorithm.

The experimental results are overwhelming. Depending on the number of variables, an improvement of up to two orders of magnitude in memory space was observed for the DIF-based algorithm in comparison to the ISC-based algorithm. An additional factor of two orders of magnitude has been achieved when the algorithm `sort0` is used as preprocessor. The algorithm `sort1` improves the effect of the algorithm `sort0` by one more order of magnitude. Altogether, an improvement by a factor of about 10^5 in memory requirements for set-based SAT solvers has been achieved. The achieved speed-up has approximately the same value.

It should be mentioned that the required time for sorting is negligible. It is very easy now to see and to understand the advantage of sorting the clauses according to the frequency of the variables in different clauses and according a general coverage. And there can be no doubt at all that in all these SAT-related problems the use of the difference and even more the use of the difference together with the sorting of the clauses will increase the efficiency in a way that is very hard to imagine.

Further experiments will be published as soon as possible. The transfer of these methods to a system of processors working in parallel also has to be considered as soon as possible.

References

[1] St. Cook: The Complexity of Theorem Proving Procedures. Proceedings of the third annual ACM symposium on Theory of computing, Shaker Heights, Ohio, United States, 1971, pp. 151-158.

[2] M. Johnson, Ch. Posthoff: TRISAT - A SAT - solver using ternary-valued logics. 14th International Workshop on Post-Binary ULSI Systems, Calgary, Canada, 2005.

[3] R.M. Karp. Complexity of computer computations. In R.E. Miller and J.W. Thatcher (editors): Reducibility Among Combinatorial Problems, New York, Plenum Press. 1972, pages 85–103.

[4] L. Levin: Universal'nye perebornye zadachi. Problemy Peredachi Informatsii 9 (3), 1973, pp. 265-266. English translation: Universal Search Problems. in B.A. Trakhtenbrot: A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms. Annals of the History of Computing 6 (4), 1984, pp. 384–400.

[5] Ch. Posthoff, B. Steinbach: Logic Functions and Equations - Binary Models for Computer Science. Springer, Dordrecht, The Netherlands, 2004.

[6] I. Wegener: Complexity Theory - Exploring the Limits of Efficient Algorithms. Springer, Dordrecht, The Netherlands, 2005.

[7] K. Zuse: The Computer My Life. Springer-Verlag, Berlin/Heidelberg, Germany, 1993. (translated from the original German edition: Der Computer Mein Lebenswerk. Springer, 1984)

[8] SATLIB - Benchmark Problems.
http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html