El-Maksoud and Abdalla *Journal of the Egyptian Mathematical Society* (2019) 27:22
https://doi.org/10.1186/s42787-019-0021-1

Journal of the Egyptian Mathematical Society

ORIGINAL RESEARCH Open Access

A novel SAT solver for the Van der Waerden numbers

Munira A. Abd El-Maksoud and Areeg Abdalla*

*Correspondence:
areeg@sci.cu.edu.eg
Department of Mathematics, Faculty of Science, Cairo, Egypt

Abstract
This paper introduces a new efficient satisfiability problem (SAT) solver, negative-literal Van der Waerden numbers SAT solver (NegVanSAT). It is a modification of the well-known SAT solver MINISAT where the constructor of the literals has been adjusted to start with the negated literals first. It reduces the calculations needed to solve a problem. The NegVanSAT is specifically designed for solving the satisfiability problem of finding Van der Waerden numbers, which are known to be very difficult to compute. Comparisons between the MINISAT and the proposed NegVanSAT show that the latter outperforms the MINISAT in finding many of them.

Keywords: Satisfiability, Van der Waerden numbers, DPLL, SAT solvers, MINISAT

Introduction

Given a Boolean formula, the problem of deciding the existence of an assignment of its variables making the formula evaluated to 1 is called the satisfiability problem (SAT) [1]. There has been many algorithms for testing the satisfiability.

The most well-known one is introduced in 1962 by "M. Davis, H. Putnam, G. Logemann, and D. Loveland" (DPLL [2]); it is considered the basis for almost all modern SAT solvers. The success of encoding many applications (like software verification [3], circuit testing [4], planning [5], and some mathematical problems) as satisfiability problems was the real reason behind the research and development of new efficient SAT solvers. Encoding a new solver from scratch is a strenuous task; instead, a researcher may modify an already existing solver to meet his needs. In this research, the solver MINISAT [6] has been modified in such a way as to be appropriate for solving the satisfiability problem of finding, the known difficult computing, Van der Waerden numbers.

The rest of this paper is organized as follows: "The satisfiability problem (SAT)" section gives a quick introduction to the satisfiability problem (SAT) and presents some logic background and notations necessary to follow the rest of the paper. "SAT and combinatorics" section explains the relationship between SAT and combinatorics and gives a SAT encoding of the problem of computing Van der Waerden numbers as an example. "SAT solvers" section presents some examples of developed solvers and compares between MINISAT, VANSAT, and NegVanSAT solvers. "Experimental results" section presents the discussion and results of the execution of NegVanSAT and MINISAT on a number of examples. And finally, "Conclusion and future work" section summarizes the work which have been done and what we intend to do in the future.

The satisfiability problem (SAT)

The following paragraphs present some logic terminologies and notations [1] necessary to follow the rest of the paper. Consider the countably infinite set of Boolean variables $X=\{x_1,x_2,\ldots\}$. These are variables which can take the two values 1 (interpreted as true) and 0 (interpreted as false). The values 1 and 0 are called "truth values."

A literal, l, is either a variable or the negation of a variable. It is referred to the variable as the positive literal and to its negation as the negative literal. Hence, a literal may have a positive or a negative polarity.

A unique unary Boolean operator, negation ($\neg$), is a mapping from 1 to 0 and vice versa. The negation of a variable, x_i, is denoted by $\neg x_i$ where the value of $\neg x_i$ is opposite that of x_i. If l is a literal, then, if l is a variable, x_i, that is, $l = x_i$, let $\bar{l} = \neg x_i$, if l is a negation of a variable, x_i, that is, $l = \neg x_i$ then let $\bar{l} = x_i$. And a binary Boolean operator is a function

$$\odot : \{0,1\} \text{ x } \{0,1\} \mapsto \{0,1\}$$

The most common binary operators are $\wedge$(And), $\vee$(Or), $\rightarrow$(Implies), $\leftrightarrow$(Equivalent), and $\oplus$(Xor). Binary Boolean operators are the building blocks of propositional expressions.

A Boolean formula (formula) is a propositional expression containing literals, Boolean operators, and parentheses whose syntax can be defined by recursion as follows:

1. Each variable is a formula.
2. If F is a formula, then $\neg F$ is also a formula.
3. If $F1$ and $F2$ are two formulas and $\odot$ is a binary Boolean operator, then $(F1 \odot F2)$ is also a formula.

A truth assignment (assignment), A, is a mapping from a finite set of Boolean variables, $S \subset X$, to the set {0,1}.

For a formula, F, any assignment, A, of values to its variables induces a value on it. The value of F is evaluated from innermost $\neg$ or parentheses out according to mappings of the Boolean operators. If the formula evaluates to 1, in this case the assignment, A, is called a solution, satisfying assignment, or a model. If such A exists, then F is satisfiable; otherwise, it is unsatisfiable. A satisfiable formula may have more than one model. Table 1 displays mappings of some common operators. A formula consisting of literals and the operator $\vee$ only is called a "disjunctive clause," which may be represented as a set of literals. For example, the clause $(x_1 \vee x_2 \vee \neg x_5)$ can be represented by the set $\{x_1,x_2,\neg x_5\}$.

A formula consisting of literals and the operator $\wedge$ only is called a "conjunctive clause." A conjunctive normal form (CNF) formula is a formula consisting of a conjunction of (two or more) disjunctive clauses. Finally, the satisfiability problem (SAT) can be stated as follows:

Given a Boolean formula, F, the question is whether F has a model, i.e., as assignment of the variables to satisfy all its clauses. In case that this assignment does not exist, another search may arise for a truth assignment that maximizes the number of satisfied clauses

Table 1 Mappings of the most common binary Boolean operators

Operator	Symbol used	Mapping
Or	$\vee$	$\{00 \mapsto 0; 01, 10, 11 \mapsto 1\}$
And	$\wedge$	$\{00, 01, 10 \mapsto 0; 11 \mapsto 1\}$
Xor	$\oplus$	$\{00, 11 \mapsto 0; 01, 10 \mapsto 1\}$
Implies	$\rightarrow$	$\{10 \mapsto 0; 00, 01, 11 \mapsto 1\}$
Equivalent	$\leftrightarrow$	$\{01, 10 \mapsto 0; 00, 11 \mapsto 1\}$

in the CNF, that is the maximum satisfiability (MaxSAT) [7]. The MaxSAT is an NP-complete problem and has been defined in other logics like Łukasiewicz logic [8].

For example, $\{(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)\}$ is a CNF formula that has the satisfying assignment $\{x_1 = 1, x_2 = 0, x_3 = 0\}$. A propositional expression in CNF is called an instance of satisfiability. The following section explains the relationship between SAT and combinatorics.

SAT and combinatorics

The research in satisfiability and combinatorics may help advance each other. On one hand, thanks to the significant efficiencies of modern SAT solvers, it became possible to encode many of the combinatorics problems as formulas and then solve their corresponding satisfiability problems. In this scenario, novel results in combinatorics are obtained.

On the other hand, combinatorics problems, as the problem of computing Van der Waerden numbers, can be utilized as a rich source of structured formulas for developing new generations of SAT solvers [9]. The next two sections introduce the Van der Waerden numbers and their SAT encoding while the solvers VANSAT and the proposed NegVanSAT are illustrated in the next section.

Van der Waerden numbers

The Van der Waerden number $w(r;t_1,t_2,\ldots,t_r)$ is the least integer m such that for every partition $\bigcup_{i=1}^{r} C_i$ of the set $\{1, 2, \ldots, m\}$, there is an index j in $\{1, 2, \ldots, r\}$ such that C_j contains an arithmetic progression (AP) of t_j terms [10]. Where, r is the number of the blocks of the partition, t_j's are the lengths of the AP's, and C_j's are the blocks of the partition.

We recall, here, that an arithmetic progression of t terms is a sequence of the form a, $a + d, \ldots, a + d(t - 1)$, where a and d are integers, $t \geq 2$, and $d > 0$ [11].

Computing Van der Waerden numbers presents an exciting but hard problem for both mathematicians and computer scientists. This challenging problem drove researchers

Table 2 Some of the known Van Der Waerden numbers

$w(r;t_1,t_2,\ldots,t_r)$		Reference
$w(2; 3, 17)$	= 279	Ahmed [10]
$w(2; 3, 18)$	= 312	Ahmed [10]
$w(2; 5, 5)$	= 178	Beeler et al. [20]
$w(2; 6, 6)$	= 1132	Kouril et al. [21]
$w(3; 2, 3, 7)$	= 55	Landman et al. [22]
$w(3; 3, 4, 4)$	= 89	Landman et al. [22]
$w(4; 2, 2, 3, 6)$	= 48	Landman et al. [22]
$w(4; 2, 2, 2, 2)$	= 76	Beeler et al. [20]
$w(4; 2, 2, 3, 7)$	= 65	Landman et al. [22]
$w(5; 2, 2, 2, 3, 3)$	= 20	Landman et al. [22]
$w(5; 2, 2, 3, 3, 3)$	= 41	Landman et al. [22]
$w(6; 2, 2, 2, 2, 4, 4)$	= 56	Ahmed [23]
$w(6; 2, 2, 2, 3, 3, 3)$	= 42	Ahmed [23]
$w(7; 2, 2, 2, 2, 2, 3, 3)$	= 24	Ahmed [23]
$w(7; 2, 2, 2, 2, 2, 3, 4)$	= 36	Ahmed [23]
$w(8; 2, 2, 2, 2, 2, 2, 3, 3)$	= 25	Ahmed [23]
$w(9; 2, 2, 2, 2, 2, 2, 2, 3, 3)$	= 28	Ahmed [23]

interested in these numbers to compute (or, at least, finding lower bounds of) many of them. Table 2 lists some of the known Van der Waerden numbers.

SAT encoding of Van der Waerden numbers

The problem of computing a Van der Waerden number, $w(r;t_1,t_2,\dots,t_r)$, can be SAT encoded as follows [12]:

Given positive integers $r,t_1,t_2,\dots,t_r$, for a positive integer, n, a CNF formula, F, can be constructed to be satisfiable if and only if $n < w(r;t_1,t_2,\dots,t_r)$. With such encodings, one can use SAT solvers to decide the satisfiability of F, and consequently, to find $w(r;t_1,t_2,\dots,t_r)$.

The algorithm used to compute Van der Waerden numbers starts with $m = r + 1$ (note that for $t_1, t_2, \dots, t_r \geq 2$, $r < w(r;t_1,t_2,\dots,t_r)$), for consecutive integers the algorithm tests whether the formula F is satisfiable. If so, it continues. If not, it returns m and terminates (the existence of Van der Waerden numbers, $w(r;t_1,t_2,\dots,t_r)$'s, guarantees the termination of the algorithm).

Constructing formulae

For a positive integer, n, consider the following two cases :

(I) $r = 2$, a formula F can be constructed with n variables as a conjunction of the following two types of clauses:

(1) $\{\neg x_a, \neg x_{a+d}, \dots, \neg x_{a+d(t_1-1)}\}$

(2) $\{x_a, x_{a+d}, \dots, x_{a+d(t_2-1)}\}$

with $a, d \geq 1$, and $a + d(t_1 - 1), a + d(t_2 - 1) \leq n$

where $x_i = 1$ encodes $i \in C_1$ and $x_i = 0$ encodes $i \in C_2$.

Clauses (1) prevent the existence of an arithmetic progression of length t_1 in C_1 and clauses (2) prevent the existence of an arithmetic progression of length t_2 in C_2 .

(II) $r > 2$, a formula F may contain nr variables, $x_{i,j}$'s, with i=1,2,...,n and j=1,2,...,r where the variable $x_{i,j}$ takes the value 1 if and only if the integer i belongs to a block C_j of a partition. In this case, F will be the conjunction of the following three types of clauses:

(1) $\{x_{i,1}, x_{i,2}, \dots, x_{i,r}\}$, for each integer i , to ensure that integer i belongs to at least one block of the partition(covering)

(2) $\{\neg x_{i,s}, \neg x_{i,t}\}$, for $1 \leq i \leq n, 1 \leq s < t \leq r$, to ensure that integer i belongs to at most one block of the partition (disjoint)

(3) $\{\neg x_{a,j}, \neg x_{a+d,j}, \dots, \neg x_{a+d(t_j-1),j}\}$, for $1 \leq j \leq r, 1 \leq a \leq n - t_j + 1$ and $1 \leq d \leq \lfloor (n-a)/(t_j-1) \rfloor$, to ensure that no arithmetic progression of length t_j in block C_j

SAT solvers

DPLL [2] was introduced in 1962 as a refinement of its earlier M. Davis and H. Putnam (DP) algorithm. Essentially, it is a (complete—"depth-first"—backtracking) search algorithm. Recently, João P. Marques-Silva and Karem A. Sakallah introduced Generic seaRch Algorithm for the Satisfiability Problem (GRASP) [13] as an extension of the DPLL [2] with learning and non-chronological backtracking. In recent decades, GRASP prompts research on conflict-driven clause learning (CDCL) solvers. A SAT solver (based on DPLL [2]) is a software, and many of SAT solvers are written in C or C^{++} [14]. Over the years, new generations of SAT solvers with significant efficiencies have been developed, some examples are multi-SAT [15], Glucose and Syrup in the SAT'17 [16], Nigma [17]and its improved versions and Glulu [18]. Figure 1 shows the different stages of converting the Van der Wearden number problem into a SAT problem until a solution is reached.

MINISAT

MINISAT [6] is a MINImal, efficient, conflict-driven clause learning (CDCL), CHAFF-like [19] SAT solver written by Eén and Sörensson. MINISAT attaches each variable with an activity and orders the variables dynamically by their activities. A variable's activity increases whenever the variable occurs in a conflict clause (all its literals have become 0); this increase is called bumping. It bumps variables with larger and larger numbers until a limit is reached (predefined number); at that point, all variable activities are scaled down. It uses a heap to sort the variables by the activity at all times [6]. In finding a solution of a given formula, MINISAT selects the unassigned (free) variable with the highest activity and tries to solve the formula first with its positive literal (with the positive polarity by default) then if failed, the solver tries its negative one.

The MINISAT has components of branching, unit propagation, and backtracking. MINISAT uses clause learning, and in modern solvers, a heuristic is developed to pick an unassigned variable and assign it either true or false, until unit propagation detects a conflict. Then, a conflict clause is constructed and added to the SAT problem and the assumption is canceled by backtracking until the conflict clause becomes unit. Finally, this unit clause is propagated and the search proceeds. The following three sections give an overview of the SAT solvers MINISAT, VANSAT, and the proposed NegVanSAT, respectively.

VANSAT

VAN der Waerden numbers SAT (VANSAT) solver [12] is a modification of MINISAT where the activity of the variable is measured by its occurrences in the not yet satisfied

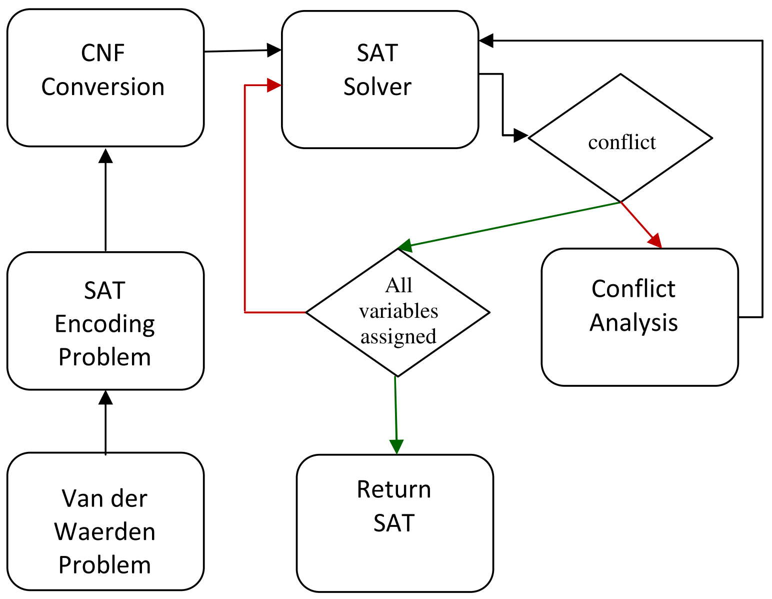

Fig. 1 Converting Van der Waerden number problem into SAT

clauses. Hence, variable activities are changed dynamically (increased and decreased) by adding and removing clauses. In other words, the strategy of VANSAT was:

I. Increasing the activity of all variables that appear in each new added clause (learnt or problem clause)

II. Decreasing the activity of all variables that were appearing in each deleted clause (where deletion of clauses occur in many situations)

Experimental results showed that the VANSAT is better in computing Van der Waerden numbers. For example, it outperformed MINISAT in computing $w(3;2,3,3)$, $w(3;2,3,5)$, and $w(4;2,2,3,3)$ in terms of the number of conflicts, decisions, restarts, propagations, and conflict literals [12].

Proposed NegVanSAT

From the above SAT encoding (of Van der Waerden numbers, $r > 2$), it has been noted that the variables occur as negative literals in all clauses except one, the first type of clauses. This observation was the motive to develop the NegVanSAT. This proposed solver is a modification of MINISAT1.14, where the constructor of the literals has been adjusted in such a way that the default literal of a variable has become the negative one. Hence, it, in contrary to MINISAT, tries solving the given formula using the negative literal of the variable before trying its positive one. And that is for the variable with the highest activity.

Experimental results

The experiments were carried out on an Intel(R)Core(TM) i3-2328M cpu@2.20 GHz machine with 4.00 GB memory.

MINISAT input format

Like most SAT solvers, MINISAT accepts input formulae written in the "Center for Discrete Mathematics and Theoretical Computer Science" (DIMACS) CNF format. A specification of the problem is written before the clauses of the formula, starts with "p" stands for problem and followed by the type of the problem, CNF; the number of variables, n; and the number of clauses, m. Each of the following, m, non-comment lines consist of a list of integers and define a clause. The integers are chosen from the set $\{1,2,\ldots,n,-1,-2,\ldots,-n\}$, appear in an arbitrary order, and is separated by spaces. Each positive literal is represented by its index while negative literals are represented by the negative values of their indices. The definition of a clause is terminated by a final value of "0." For example, the CNF formula:

Table 3 Number of conflicts

$w(r;t_1,t_2,\ldots,t_r)$	NegVanSAT	MINISAT
W1:$w(3;2,3,5)$ for $n=31$	381	743
W2:$w(4;2,2,3,3)$ for $n=12$	4	9
W3:$w(4;2,2,3,4)$ for $n=25$	9198	17596
W4:$w(4;2,2,3,4)$ for $n=20$	1	86
W5:$w(5;2,2,2,3,5)$ for $n=30$	9	170
W6:$w(5;2,2,2,3,5)$ for $n=44$	1423072	1978087
W7:$w(5;2,2,2,3,3)$ for $n=15$	0	18

Table 4 Number of decisions

$w(r;t_1,t_2,\ldots,t_r)$	NegVanSAT	MINISAT
W1:$w(3;2,3,5)$ for $n=31$	415	1006
W2:$w(4;2,2,3,3)$ for $n=12$	12	27
W3:$w(4;2,2,3,4)$ for $n=25$	10199	25260
W4:$w(4;2,2,3,4)$ for $n=20$	14	105
W5:$w(5;2,2,2,3,5)$ for $n=30$	32	278
W6:$w(5;2,2,2,3,5)$ for $n=44$	1542234	2835250
W7:$w(5;2,2,2,3,3)$ for $n=15$	13	36

$(\neg x_7 \lor x_3 \lor x_2) \land (x_1 \lor x_2 \lor \neg x_5) \land (x_4 \lor \neg x_6 \lor x_8)$ is coded as:

```
p cnf 8 3
-7 3 2 0
1 2 -5 0
4 -6 8 0
```

Codes are written to translate the SAT encoding of Van der Waerden numbers to the "DIMACS CNF" format.

Assessment methods

The common measures to compare SAT solvers are the following [6]:

◇ Number of conflicts (C)

A conflict occurs when all literals of a clause have become 0. During search, the number of conflicts increases by 1 whenever a conflict occurs.

◇ Number of decisions (D)

MINISAT starts its search by selecting (heuristically) an unassigned variable and assigning it a value, this process is called a "decision." Every time a new variable decision is made, one is added to the decisions counter.

◇ Number of restarts (R)

MINISAT begins its search procedure with a bound on the number of conflicts. If reached, the solver will be forced to restart and one is added to the number of restarts.

◇ Number of propagations (P)

First, let us look at how MINISAT implements propagations. For each literal, l, a list of clauses is kept; these are the clauses which will be inspected when l becomes 1, such lists are referred to as " watcher lists." For each clause, C, two of its unassigned literals l_1 and l_2

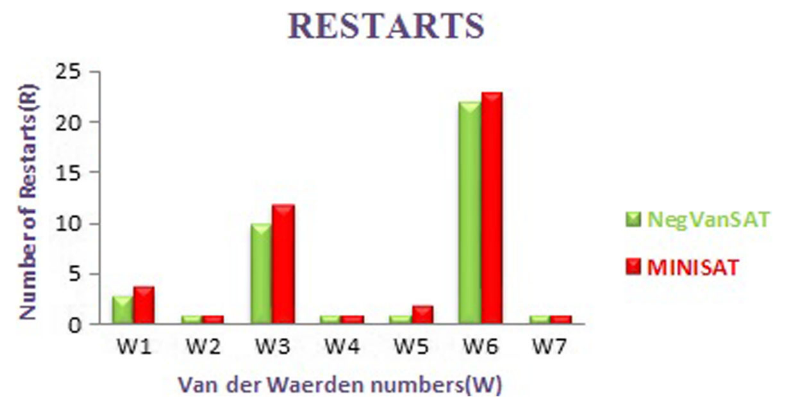

Fig. 2 NegVanSAT Vs MINISAT in the restarts

Table 5 Number of restarts

$w(r;t_1,t_2,\ldots,t_r)$	NegVanSAT	MINISAT
W1:$w(3; 2, 3, 5)$ for $n = 31$	3	4
W2:$w(4; 2, 2, 3, 3)$ for $n = 12$	1	1
W3:$w(4; 2, 2, 3, 4)$ for $n = 25$	10	12
W4:$w(4; 2, 2, 3, 4)$ for $n = 20$	1	1
W5:$w(5; 2, 2, 2, 3, 5)$ for $n = 30$	1	2
W6:$w(5; 2, 2, 2, 3, 5)$ for $n = 44$	22	23
W7:$w(5; 2, 2, 2, 3, 3)$ for $n = 15$	1	1

are selected and references to C are added to the lists of $\bar{l_1}$ and $\bar{l_2}$, respectively. If a clause is found in a watcher list during propagation of a literal, its propagate method is called and executed.

MINISAT mantains a queue called a "propagation queue" to keep the literals which have to be propagated upon propagation process. On propagations, all of the enqueued literals are propagated and a propagations counter is increased by one.

⋄ Number of conflict literals (Cl)

When a conflict takes place, it is analyzed and a so-called "learnt clause" is produced and added to the clauses for prohibiting the variable assignments which leads to such a conflict. During analysis, the number of literals of learnt clauses is counted here.

Evaluation of results and discussions

The following tables show a comparison between the results of NegVanSAT and MINISAT solvers for a number of Van der Waerden numbers where NegVanSAT was better. Table 3 compares the number of conflicts in both of NegVanSAT and MINISAT.

It is clear that NegVanSAT greatly outweighed MINISAT in all the listed numbers. For the first three numbers, NegVanSAT gives about half number of conflicts, while in W7, it has reached 0 conflicts and that will recall less number of restarts which reflects in better memory usage.

Table 4 compares the number of decisions that were made by both NegVanSAT and MINISAT. It is clear that, for all the listed Van der Waerden numbers, the proposed SAT requires less number of decisions in the search for a solution.

Table 2 compares the number of restarts in both of NegVanSAT and MINISAT. For $w(4; 2, 2, 3, 3)$ when $n = 12$, $w(4; 2, 2, 3, 4)$ when $n = 20$, and $w(5; 2, 2, 2, 3, 3)$ when $n = 15$ NegVanSAT had the same number of restarts, yet in the rest, it was better. It showed that the NegVanSAT was overall at least as good as MINISAT if not better. Figure 2 shows the NegVanSAT was better than the MINISAT in computing four of the numbers listed in Table 5 and had the same number of restarts in three of them. Which means that it

Table 6 Number of propagations

$w(r;t_1,t_2,\ldots,t_r)$	NegVanSAT	MINISAT
W1:$w(3; 2, 3, 5)$ for $n = 31$	9729	23063
W2:$w(4; 2, 2, 3, 3)$ for $n = 12$	94	167
W3:$w(4; 2, 2, 3, 4)$ for $n = 25$	283096	545502
W4:$w(4; 2, 2, 3, 4)$ for $n = 20$	96	1833
W5:$w(5; 2, 2, 2, 3, 5)$ for $n = 30$	293	5564
W6:$w(5; 2, 2, 2, 3, 5)$ for $n = 44$	68616554	86707553
W7:$w(5; 2, 2, 2, 3, 3)$ for $n= 15$	75	312

Table 7 Number of conflict literals

$w(r;t_1,t_2,\ldots,t_r)$	NegVanSAT	MINISAT
W1:$w(3; 2, 3, 5)$ for $n = 31$	4395	7578
W2:$w(4; 2, 2, 3, 3)$ for $n = 12$	37	128
W3:$w(4; 2, 2, 3, 4)$ for $n = 25$	89482	194914
W4:$w(4; 2, 2, 3, 4)$ for $n = 20$	12	1607
W5:$w(5; 2, 2, 2, 3, 5)$ for $n = 30$	125	4446
W6:$w(5; 2, 2, 2, 3, 5)$ for $n = 44$	33518252	57269153
W7:$w(5; 2, 2, 2, 3, 3)$ for $n = 15$	0	342

restarts the search at most as the MINISAT if not less. Table 6 compares the number of Propagations in both of NegVanSAT and MINISAT where NegVanSAT was markedly better than MINISAT. Table 7 compares the number of conflict literals in both of NegVanSAT and MINISAT. As was expected, from the comparison of the number of conflicts, the number of conflict literals counted during the work of NegVanSAT is much less than in MINISAT. which again was a natural result of starting with the negative literal first in the search. NegVanSAT always shows superiority over MINISAT in computing Van der Werden numbers. Figure 3 shows an example of that in computing W2:$w(4; 2, 2, 3, 3)$ for $n = 12$ in the measures C, D, P, and Cl.

For all the above measures, the NegVanSAT has proved its superiority over the MINISAT. Which has been a nature result of solving the given formula using the negative literal first. Since it has been noted that the variables occur as negative literals in all clauses except one.

Conclusion and future work

This paper introduces a new SAT solver, NegVanSAT, based on the well-known SAT solver MINISAT. NegVanSAT is designed specifically to compute Van der Waerden numbers. It is a modification of the MINISAT, where the constructor of the literals has been adjusted in such a way that the default literal of a variable has become the negative one. Experiments showed that the NegVanSAT outperformed the MINISAT in computing many of the Van der Waerden numbers.

In the future, we aim to find new Van der Waerden numbers using the new solver. Also we aim to study the solver complexity but this requires a better computing facilities.

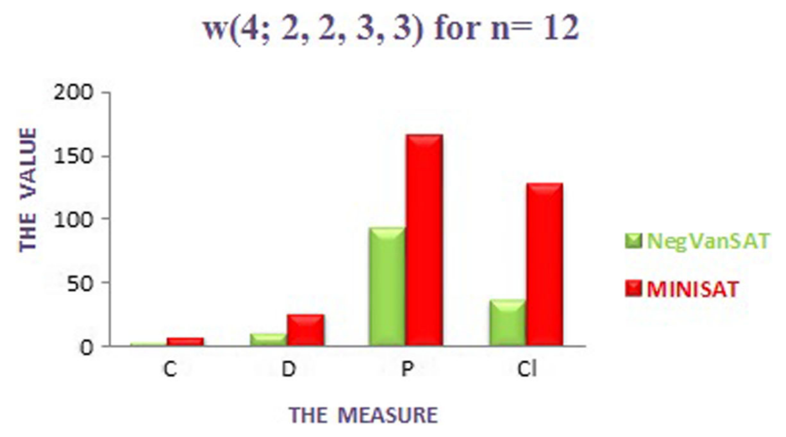

Fig. 3 An example of the superiority of NegVanSAT

Abbreviations
AP: Arithmetic progression; CDCL: Conflict-driven clause learning; CNF: Conjunctive normal form; DPLL: Algorithm by M. Davis, H. Putnam, G. Logemann, and D. Loveland; GRASP: Generic seaRch algorithm for the satisfiability problem; MaxSAT: Maximum satisfiability; NegVanSAT: Negative-literal Van der Waerden numbers SAT solver; SAT: Satisfiability problem; VANSAT: VAN der Waerden numbers SAT

Acknowledgements
The authors are grateful to the referees for their valuable suggestions.

Authors' contributions
Both authors jointly worked on the results, and they read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Received: 28 September 2018 Accepted: 22 November 2018
Published online: 19 July 2019

References

1. Franco, J., Weaver, S.: Algorithms for the Satisfiability Problem, Handbook of Combinatorial Optimization. pp. 311–454. Springer, New York (2013)
2. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM. **5**(7), 394–397 (1962)
3. Jackson, D., Vaziri, M.: Finding bugs with a constraint solver. International Symposium on Software Testing and Analysis, Portland, Oregon (ISSTA). 14–25 (2000)
4. Stephan, P., Brayton, R. K., Sangiovanni-Vincentelli, A. L.: Combinational test generation using satisfiability. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **15**, 1167–1176 (1996)
5. Kautz, H., Selman, B.: Planning as satisfiability. **92**, 359–363 (1992). European Conference on Artificial Intelligence
6. Eén, N., Sörensson, N. S.: Springer: An Extensible SAT-solver. In: Theory and Applications of Satisfiability Testing, pp. 333–336. Springer, (2004)
7. El Halaby, M.: Solving MaxSAT by successive calls to a SAT solver. Lect. Notes Netw. Syst. **15** (2018)
8. El Halaby, M., Abdalla, A.: Fuzzy Maximum Satisfiability. In: Proceedings of the 10th International Conference on Informatics and Systems (INFOS), Cairo, Egypt. ACM, pp. 50–55, (2016)
9. Dransfield, M. R., Liu, L., Marek, V. W., Truszczynski, M.: Satisfiability and computing van der Waerden numbers. Electron. J. Comb. **11**(1), #R41 (2004)
10. Ahmed, T.: Two new Van der Waerden numbers: w(2; 3, 17) and w(2; 3,18). Integers. **10**, 369–377 (2010)
11. Xiu, B., Li, G., Liang, M., Xu, X.: A set-coloring generalization of Van der Waerden numbers. J. Comput. Theor. Nanosci. **11**, 2431–2436 (2014)
12. El-Maksoud, M. A., Abdalla, A.: An improvement and implementation of the dpll satisfiability algorithm. In: The 52^{nd} Annual Conference on Statistics, Computer Sciences and Operation Research, pp. 25–27, Cairo, (2017)
13. Marques-Silva, J. P., Sakallah, K. A.: GRASP: a new search algorithm for satisfiability. IEEE Trans. Comput. **48**, 506–521 (1999)
14. Zhang, L., Malik, S.: The quest for efficient boolean satisfiability solvers. In: Proceedings of the 14^{th} International Conference on Computer Aided Verification, pp. 17–36. Springer-Verlag, London, (2002)
15. Siddiqi, S., Huang, J.: multi-SAT: an adaptive SAT solver. In: Proceedings of SAT Competition 2016 , Solver and Benchmark Descriptions, p. 54, (2016). https://pdfs.semanticscholar.org/dec3/ff8cf104d347dadc85e4fb4f8f13a835cb62.pdf
16. Audemard, G., Simon, L.: Glucose and syrup in the SAT'17. In: Proceedings of SAT Competition 2017 , Solver and Benchmark Descriptions, pp. 16–17, (2017). http://hdl.handle.net/10138/224324
17. Jiang, C., Zhang, T.: Nigma: a partial backtracking SAT solver. In: Proceedings of SAT Competition 2013 , Solver and Benchmark Descriptions, pp. 62–63, (2013). http://hdl.handle.net/10138/40026
18. Zha, A.: Glulu. In: Proceedings of SAT Competition 2017, Solver and Benchmark Descriptions, p. 18, (2017). http://hdl.handle.net/10138/224324
19. Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38^{th} conference on Design Automation, pp. 530–535, New York, (2001)
20. Beeler, M. D., O Neil, P. E.: Some new Van der Waerden numbers. Discret. Math. **28(2)**, 135–146 (1979)
21. Kouril, M., Paul, J. L.: The Van der Waerden number w(2, 6) is 1132. Exp. Math. **17(1)**, 53–61 (2008)
22. Landman, B., Robertson, A., Culver, C.: Some new exact Van der Waerden numbers. Integers Electron. J. Comb. Number Theory. **52**(2), 1–11 (2005)
23. Ahmed, T.: Some new Van der Waerden numbers and some Van der Waerden-type numbers. Integers. **9**, 65–76 (2009)