SAT-Problems - New Findings

CHRISTIAN POSTHOFF
Department of Mathematics & Computer Science
The University of The West Indies
St. Augustine Campus
TRINIDAD & TOBAGO
http://www.uwi.tt/fsa/dmcs/computerscience/staff/cposthoff.html

BERND STEINBACH
Institute of Computer Science
Freiberg University of Mining and Technology
Bernhard-von-Cotta-Str. 2, D-09596 Freiberg
GERMANY
http://www.informatik.tu-freiberg.de/index.html

Abstract: - SAT-problems in general and 3-SAT problems in particular are very important and interesting NP-complete problems with many applications in different areas. In several previous papers, before all in [2] and in [5], 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. The approach does not use any of the classical search methods, it uses more constructive ways to build possible solutions on the basis of partial solutions which are more or less easy to find. Experiments showed that this approach is very competitive and efficient. After the parallelism of the solution process has been established on the register level, i.e. related to the existing hardware, it could also be shown that it is very promising and efficient to extend the ideas and concepts to the use of several processors working in parallel (see, for instance, [8]). This paper now presents some refinements of the existing approaches with an overwhelming and very surprising increase of the efficiency of the algorithms and implementations. The presentation relates to one processor, the transfer to a multi-processor system will be presented as soon as possible.

Key-Words: - SAT-problems, 3-SAT, bit-parallel vector operations, intersection, difference, sorting, efficiency.

1 Introduction

Boolean problems combine practical and theoretical aspects in a special way and can be considered as one of the most important foundations of Computer Science and its applications. Besides the theory of Boolean Algebras and the Boolean Differential Calculus [5], certain Boolean problems appear as key in complexity theory [7]. 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 transformation. And this is also the bridge to many applications related to the design of digital devices: the reduction of a real-world problem to binary representations and the processing of these representations. Any improvement of the basic algorithms will also improve the efficiency of algorithms dealing with many applications.

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's Theorem. NP-complete problems are the most interesting problems in NP. In the context of the practical importance mentioned above we present in this paper some new findings that improve the already efficient algorithms further and allow the solution of many more and larger problems in a very short or acceptable time.

2 Preliminaries

In order to make this note independently readable, we summarize the concepts that have been used before. 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 can be assigned to its variables in a way that makes the expression true or equal to 1 (0 and 1 are used in hardware-related considerations, **false** and **true** more in applications related to logics). 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 a conjunctive form having three or more variables in its clauses. In the special case of the 3-SAT problem each clause includes three literals.

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 eight solutions.
- If $x_2 = 0$, then $\overline{x}_3 = 1$, i. e. $x_3 = 0$ gives four more solutions that are not covered by the previous case and described by $(-00-)$.
- If $x_2 = 0$ and $x_3 = 1$, then x_4 must be equal to 1. Thus, (-011) is another set of two solutions.
- There are no further solutions.

In this way we find three ternary vectors for the solution of $C = 1$, and they describe **disjoint** solution sets. This simple mechanism avoids completely the search for double solutions, one of the crucial points in this area. The $-$ in the first position indicates that in the context of the problem a variable x_1 is assumed to play a role.

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 the intersection of the respective solution sets.

The intersection will be computed according to Table 1 which has to be applied in each component. The $\emptyset$ 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 intersection is empty if

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

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

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

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

Hint: $\oplus$ indicates the exclusive-or. Hence, by using some very fast and very simple bit vector operations (available on the hardware level), 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 The overlap of disjunctions

In the following we will see that the overlap of variables in different disjunctions will be important for the growth of intermediate results. The overlap in one variable is the most important case, the overlap in two or even three variables does not need special consideration, it shows the same consequences.

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

$$\Omega_1(n) = n \begin{pmatrix} n-1 \\ 2 \end{pmatrix} \begin{pmatrix} n-3 \\ 2 \end{pmatrix} =$$

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

possible pairs of disjunctions overlapping in one and only one variable.

Including also the overlap in two or three variables, we can find out by a detailed analysis that for $n = 10$ 70.83% of the pairs of disjunctions show an overlap, and for $n = 20$ still 40.26% of the pairs of disjunctions show an overlap.

Now we will explore the effect of the overlap.

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

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

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

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

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

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

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

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

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

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

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

This case can be reduced to the previous case using the substitution $\bar{c} = x$ 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 orthogonal representation 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.

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

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

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

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

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

The other overlap possibilities (overlap in two or even three variables) including the consideration of different possibilities of negated or non-negated variables also result in three or four vectors in the intersection.

A very efficient strategy can now 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.

One example of 50 variables and 218 clauses, for instance, showed the following frequencies for $x_1, ..., x_{50}$: x_{14} : 22, x_{49} : 19, x_3 : 16, x_{22} : 11, ... Hence, the clauses have been sorted according to these frequencies: first all clauses with x_{14} followed by all clauses containing x_{49} etc. The experimental results showed an enormous increase of the efficiency (see below).

4 Intersection versus Set Difference

Now we consider the intersection as the main operation of the approach. Let us look at an intermediate result such as

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

and let $D = (c \vee d \vee e)$ be the next disjunction to be considered. According to our previous considerations we get the solution set for $D = 1$ and the intersection $A \cap D$ as follows: -

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

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

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

Now we change the point of view. We consider vectors that must be excluded from the space of possible solutions: $c = 0, d = 0, e = 0$ will always result in $D = 0$, these vectors have to be deleted from A, we must determine $A \setminus (- - 000)$. This can be done in a very elegant and efficient way using again the orthogonality of ternary vectors. We take, for instance, the first vector of A, $(011 - -)$, and see immediately that $(011 - -) \cap (- - 000) = \emptyset$. The same applies to the second and the third vector: these three vectors can 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, and the representation of the matrix for $A \cap D = A \setminus (- - 000)$ only needs 5 lines instead of 8:

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

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

The efficiency of this 'tiny' step will be seen when we look at the experimental results.

5 Some Experimental Results and Conclusions

The experiments have been very interesting and successful, a broader presentation will be given at the Conference as well as on the Internet and by more detailed and comprehensive publications. In order to be comparable, two examples given in [10] have been chosen at random. 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.
Maximum at clause: i=57.
Computing time: 92,312 msec.

Intersection and Sorting
Maximum number of intermediate clauses: 63,368,192.
Memory overflow at clause $i = 75$.
Computing time until overflow: 45,375 msec.

Difference and Sorting
Maximum number of intermediate clauses: 525,660.
Maximum at clause: i=78.
Computing time: 843 msec.

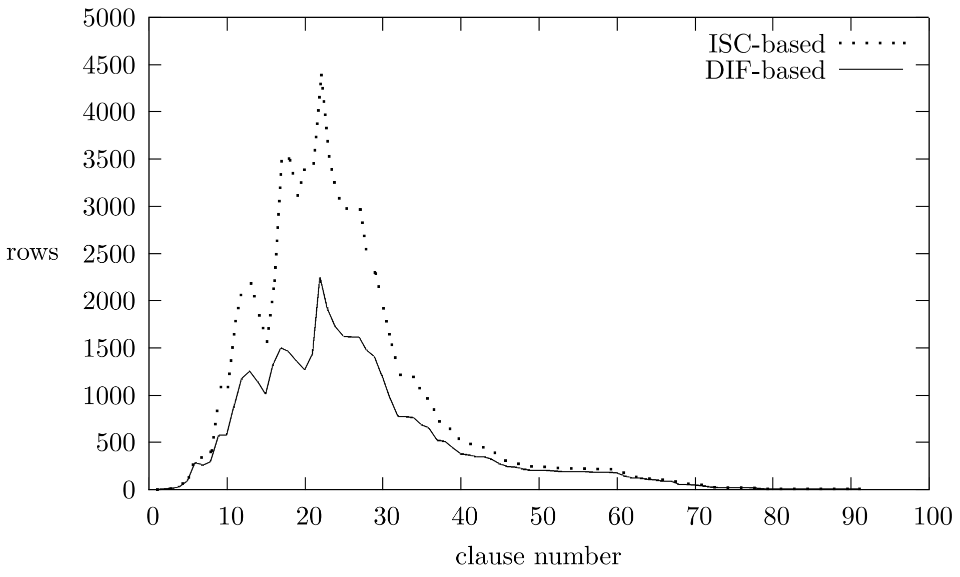


Figure 1: Comparison of the number of ternary vectors in the intermediate TVL using the ISC-based and the DIF-based algorithm for the solution of the SAT benchmark uf20-01 that depends on 20 variables and 91 clauses, on a linear scale of the rows.

Figure 2 shows the size of the intermediate TVL after the computation of each clause for both algorithms a) without sorting and b) including sorting using a logarithmic scale.

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 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. The sorting alone created in this example an improvement expressed by a factor of 110 in computing time and 166 in memory space. 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 Search) 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.

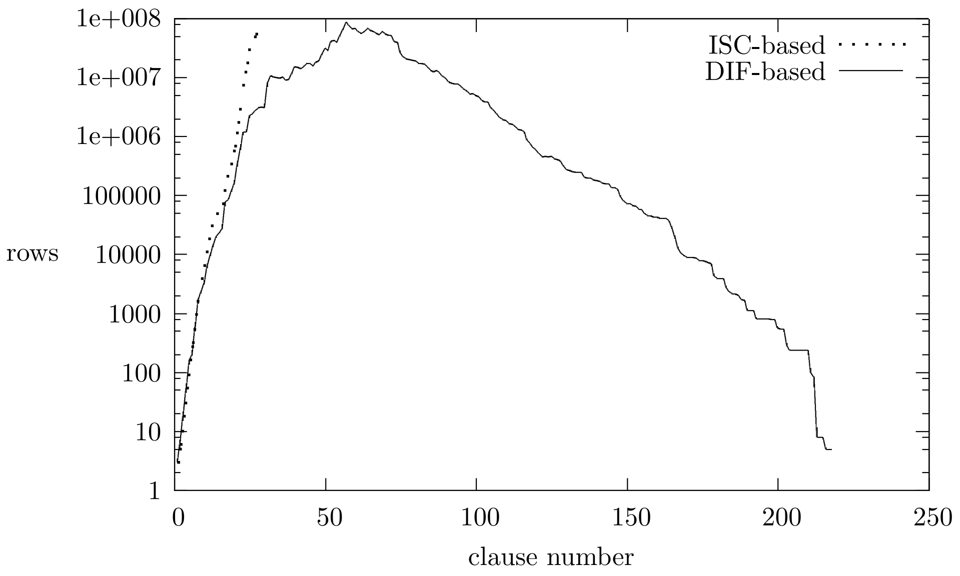
[6] B. Steinbach, N. Kümmling, Effiziente Lösung hochdimensionaler Boolescher Probleme mittels XBOOLE auf Transputer. *Transputeranwendertreffen TAT'90*, Aachen, Germany, 1990.

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

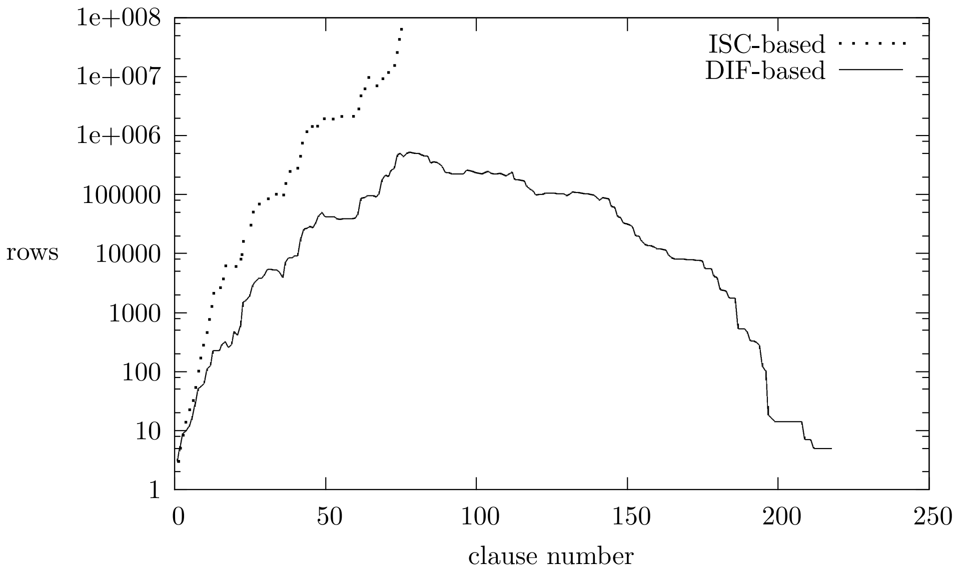
[8] Ch. Posthoff, B. Steinbach, A Multi-Processor Approach to SAT-Problems. *7th International Workshop on Boolean Problems*, 19th - 20th of September 2006, Freiberg University of Mining and Technology, Freiberg, Germany, 2006.

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

[10] *SATLIB - Benchmark Problems.* http://www.cs.ubc.ca/~hoos/SATLIB/



a)



b)

Figure 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: a) without soting, b) with sorting.