

A Partition Rule for SAT Solvers: The Multiple Partition Rule (MPR)

Juan Segura-Salazar¹, Juan Frausto-Solís²

¹Genomic Science Center, UNAM, P.O. Box 565-A, Av. Universidad s/n, C.P. 62210, Cuernavaca Morelos, México.

²ITESM, Cuernavaca Campus, A.P. 99-C, Paseo de la Reforma 182-A, C.P. 62589, Cuernavaca Morelos, México.

Abstract

We propose a new partition rule for *DPLL*-based *SAT* Solvers. Most of the complete *SAT* solvers usually are based on Davis, Logemann and Loveland (*DPLL*) rules. One most *DPLL* rule actually used in the modern algorithms is the Classical Partition Rule (*CPR*), that divides the problem into sub-problems (resolvents) and thereby it finds a solution through a decision tree. In this paper a new partition rule named Multiple Partition Rule (*MPR*) is presented. *MPR* generates a new decision tree according to clauses instead *CPR* which generates a decision tree according to variables. *MPR* can be used for developing new *SAT*'s algorithms and to improve existence ones that use *CPR*. Experimental results comparing *MPR* versus *CPR* show that using *MPR* makes more efficient solutions than *CPR*.

Keywords: Propositional satisfiability problem, *SAT*, Np-Complete, Algorithms.

1. Introduction

All Since Cook proved that the propositional satisfiability problem (*SAT*) [1] was the first problem shown to be NP-Complete [2], many other problems have been considered in this category [3]. However, *SAT* remains at the core of this classification due: i) to its simple representation (this does not mean that *SAT* can be solved easily), and ii) it has been shown that many other NP-Complete problems can be transformed to a *SAT* instance in polynomial time [4].

Owing to its importance, many *SAT* algorithms have been developed based either on local search (incomplete) or backtrack search (complete) [5]-[6]. It is striking that most popular and useful complete algorithms for solving *SAT* are based on Davis, Logemann and Loveland algorithm (*DPLL*) rules [7]. *DPLL* [8] is a modified version of the algorithm proposed by Davis & Putnam in 1960 [7]. The *DPLL* corresponds to a backtrack search; the Boolean Constraint Propagation (*BCP*) use the Classical

Partition Rule (*CPR*) [9] to make a decision tree to order the solution and apply backtracking. Modern *SAT* solvers like *zchaff* [10] and *Grasp* [11] have efficient *BCP* engines for detection of unit clauses, propagation unit literals and conflict detection. This solvers improve the basic backtrack search of *DPLL* implementing a non-chronology backtrack and conflict analysis, but using the basic *BCP* to partition the problem. In this sense, given the importance of *CPR* in the execution of *SAT* solvers, it is very important to create alternative rules in order to get more efficient *SAT* algorithms based in *CPR* rule.

This paper introduces a new partition rule named Multiple Partition Rule (*MPR*) which focuses on solving clauses in contrast to *CPR* that solves variables, and generates a new decision tree in the *SAT* propagation process. To demonstrate the efficiency of *MPR*, it was implemented on an algorithm named *MPR_Solver* which involves a different partition mechanism, and *CRP* it was implemented on an algorithm named *CPR_Solver*. Results show that *MPR*'s efficiency outperforms *CPR*'s efficiency in fifty percent. The paper organization is as follows: in the next section basic definitions are provided, in section 3 the Classical Partition Rule (*CRP*) [7] is analyzed and discussed. In section 4, the Multiple Partition Rule (*MPR*) is presented and its application in a *SAT* algorithm named *MPR_Solver* to compare with *CPR_Solver* is discussed in Section 5.

2. Basic Definitions

To represent an instance of *SAT*, the next elements are required:

- A set of variables which can be false (0) or true (1), $\beta = x_1, \dots, x_n$
- A set of literals L . A literal is a variable that can be denied or not denied, this is:

$$\exists x \in \beta: (L = x) \vee (L = \sim x) \quad (1)$$

- A set of clauses, $C = c_1, \dots, c_m$ where each one is a disjunction of literals, this is:

$$c_h = \bigvee_{1 \leq i \leq n} x_i^j, -1 \leq j \leq 1, 1 \leq h \leq m \quad (2)$$

$$\text{Where: } j = \begin{cases} x_i^{-1} = \{\} \\ x_i^0 = x_i \\ x_i^1 = \sim x_i \end{cases}$$

- A boolean formula in CNF, this is:

$$\varphi = \bigwedge_{1 \leq h \leq m} c_h \quad (3)$$

The CNF formula is also represented by a set of clauses, rather than as a conjunction. For example, an alternative way to represent φ is as the set $\{C_1, \dots, C_m\}$. In this sense, a formal definition of *SAT* is given next:

Definition 1. Find some true assignment for the variables, such that the formula φ is true; otherwise demonstrate that an assignment of values does not exist to evaluate φ as true.

Many *SAT* instances are limited to the length k ; such instances are denoted *k-SAT*. The most common instances of special interest are *2-SAT* and *3-SAT*. It has been proved that for $k = 2$ such instances are solved in polynomial time [12]-[13] while for $k = 3$ the instances are *NP*-complete [2]. In this paper, we use *3-SAT* instances in order to prove the *MPR* approach.

The propagation for variable x is denoted by $\varphi[x]$, that means:

1. If $x = 1$, remove $\sim x$ from φ 's clauses and remove clauses containing x from φ .
2. If $x = 0$, remove x from φ 's clauses and remove clauses containing $\sim x$ from φ .

If φ contains no clauses, then $\varphi = 1$ (true). The *SAT* problem represented by φ is satisfiable if there exist an assignment λ , such that $\varphi = 1$ under λ , otherwise it is unsatisfiable.

An empty clause denoted by \square is a clause that does not have literals and always it's unsatisfiable.

3. Classical Partition Rule (CPR)

The *DPPL* [7] is backtrack search algorithm. Two rules are the base for this algorithm:

1. Classic Partition Rule (*CRP*), called also splitting Literal Rule:

If the set φ can be expressed in the way:

$$(x_{1,1} \vee x_{1,2} \vee \dots \vee x_{1,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \quad (4)$$

where $x_{i,j}$ is the variable j of the clause i , then the following set is obtained:

$$\begin{aligned} \varphi_1 &= x_{1,1} \wedge (x_{1,1} \vee x_{1,2} \vee \dots \vee x_{1,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \\ \varphi_2 &= \sim x_{1,1} \wedge (x_{1,1} \vee x_{1,2} \vee \dots \vee x_{1,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \\ &\vdots \\ \varphi_{k-1} &= \sim x_{m,k} \wedge (x_{1,1} \vee x_{1,2} \vee \dots \vee x_{1,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \\ \varphi_k &= \sim x_{m,k} \wedge (x_{1,1} \vee x_{1,2} \vee \dots \vee x_{1,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \end{aligned} \quad (5)$$

φ is unsatisfiable if and only if $(\varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_k)$ is unsatisfiable; that is, if all are unsatisfiable.

2. Unit Clause Rule (*UCR*):

If exist a clause $c_h = x_n$ in φ , then $\varphi[x_n]$, if exist a clause $c_h = \sim x_n$ in φ , then $\varphi[\sim x_n]$.

CPR to divide the problem into two sets according to a selected variable; one set is evaluated with a true value and the other with a false value. The resulting sets are solved searching for one that is satisfiable or both are unsatisfiable. *CPR* also orders the solution search by creating a binary decision tree, which eliminates repetition of partial assignments proven previously. A possible decision tree formed by *CRP* is shown in the Fig. 1.

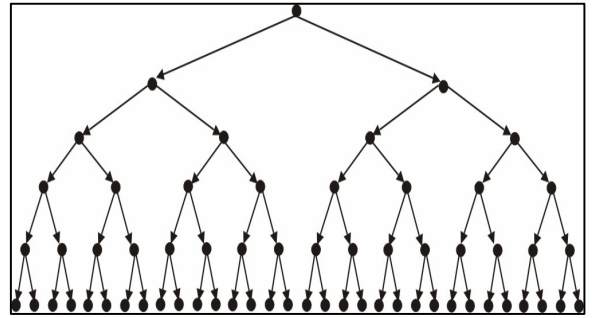


Fig. 1: Possible decision tree formed by the *CRP*.

In Fig. 1, each node represents a selected variable to be assigned, and each arch represents the propagation of this variable. The numbers of operations are counted when the selected variable is propagated.

In the case of the possible decision tree for *CRP* shown in Fig. 1, the number of arches in the last level

is 2^n . The first level involves the derivation of two arches, which then form two arches for the next level; the process continues in this way until the last level. Hereof, the total number of arches is:

$$\sum_{i=1}^n 2^i = 2^{n+1} - 2 \quad (6)$$

Modern complete *SAT* solvers implement the *CRP* on the following way: A partial assignment ω , initialized empty, is maintained. A ω is extended by a *BCP* [14]; then, if the input ϕ is neither 1 nor 0 under ω , the *CRP* is used, the satisfiability of ϕ is recursively checked under $\{\omega, x\}$ and then under $\{\omega, \sim x\}$, as required. The x is a selected variable that is picked using some decision heuristic.

4. The Multiple Partition Rule (MPR)

In a *SAT* instance, clauses are formed by several variables joined with the disjunction operator (OR). It means that at least one variable in a clause needs to have a true value (1) to make its respective clause true. It also implicates not assigning the values that transforms the clause into an empty one without affecting the values that are assigned to the other variables. For example, the following is an examination of the clause $c_i = (x_1 \vee x_2 \vee x_3)$ of an instance ϕ with a literal set $\{x_1, x_2, x_3, x_4, x_5\}$. The clause c_i indicates that any one of x_1, x_2 or x_3 should take a true value to make this clause true; it also indicates that the assignment $\{\sim x_1, \sim x_2, \sim x_3\}$ with any of the other variables is invalid. The set of invalid assignments is shown in the Fig. 2.

$$\begin{aligned} & \{ \sim x_1, \sim x_2, \sim x_3, x_4 \}, \\ & \{ \sim x_1, \sim x_2, \sim x_3, x_4, x_5 \}, \\ & \{ \sim x_1, \sim x_2, \sim x_3, x_4, \sim x_5 \}, \\ & \{ \sim x_1, \sim x_2, \sim x_3, \sim x_4 \}, \\ & \{ \sim x_1, \sim x_2, \sim x_3, \sim x_4, x_5 \}, \\ & \{ \sim x_1, \sim x_2, \sim x_3, \sim x_4, \sim x_5 \} \\ & = \{ \sim x_1, \sim x_2, \sim x_3, \sim \} \end{aligned}$$

Fig. 2: Set of invalid assignments, clause $(x_1 \vee x_2 \vee x_3)$

In the case of the decision tree of *CRP* shown in Fig. 1, it accepts this invalid assignment despite the *CRP* is focused on the selected variable. In this sense, the *CRP* procedure (when the variable x_1 has been selected to be propagated, and the variables x_2, x_3, x_4 , and x_5 remain in the clause) can be expressed by:

$$\text{CRP. If } ((\phi[\omega, x_1] \{x_2, x_3, x_4, x_5\}) = 0) \text{ Then} \quad (7) \\ (\phi[\omega, \sim x_1] \{x_2, x_3, x_4, x_5\})$$

From (7) it is clear that the *CPR* procedure divides the instance in two parts looking for the solution in any one of them: one assigning a true value to the selected variable (x_i) and other with a false value ($\sim x_i$). It is also clear that invalid assignments will be taken into account in the propagation procedure.

We propose a new partition rule (*MPR*) which focuses on the clause. *MPR* consider a clause by transforming the partition process according to the values that will be assigned to its variables avoiding invalid assignments. The *MPR* function is formed in the following way:

$$\begin{aligned} \text{MPR. If } ((\phi[\omega, x_1] \{x_2, x_3, x_4, x_5\}) = 0) \text{ Then} \quad (8) \\ \text{If } ((\phi[\omega, \sim x_1, x_2] \{x_3, x_4, x_5\}) = 0) \text{ Then} \\ (\phi[\omega, \sim x_1, \sim x_2, x_3] \{x_4, x_5\}) = 0) \end{aligned}$$

The valid values for the clause $(x_1 \vee x_2 \vee x_3)$ are $\{x_1, *\}$, $\{\sim x_1, x_2, *\}$ and $\{\sim x_1, \sim x_2, x_3, *\}$. Using this function repeatedly, the multiple partition rule is intuitively designed as follows:

Multiple Partition Rule (MPR): If the set ϕ can be expressed in the way:

$$(x_{1,1} \vee x_{1,2} \vee \dots \vee x_{1,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \quad (9)$$

where $x_{i,j}$ is the variable j of the clause i , then the following set is obtained:

$$\begin{aligned} \phi_1 &= x_{1,1} \wedge (x_{2,1} \vee x_{2,2} \vee \dots \vee x_{2,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \\ \phi_2 &= \sim x_{1,1} \wedge x_{1,2} \wedge (x_{2,1} \vee x_{2,2} \vee \dots \vee x_{2,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \\ &\vdots \\ \phi_2 &= \sim x_{1,1} \wedge \sim x_{1,2} \wedge \dots \wedge x_{1,k} \wedge (x_{2,1} \vee x_{2,2} \vee \dots \vee x_{2,k}) \wedge \dots \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,k}) \end{aligned} \quad (10)$$

ϕ is unsatisfiable if and only if $(\phi_1 \vee \phi_2 \vee \dots \vee \phi_k)$ is unsatisfiable; that is, if all are unsatisfiable.

With this new rule, the variable to be selected considers the result of a previous assignment in the clause. When *MPR* gives unsatisfiable in the evaluation process for a specific value of a selected variable x_i , *MPR* now considers this variable with the contrary value. After x_i has been assigned with true and false, *MPR* proceeds to select another variable to spread. This process is shown in Fig. 3.

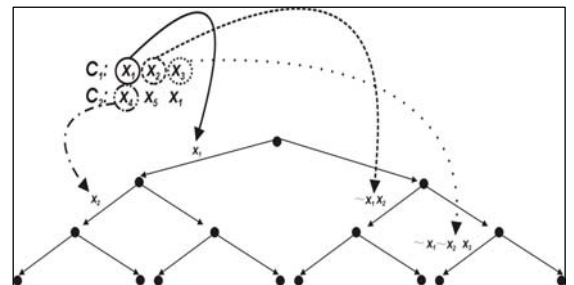


Fig. 3: Possible decision tree formed by *MPR*.

The application of this rule is an improvement since the number of selections decreases. In the decision tree formed by *MPR* (Fig. 3), the right arcs fix the value, and not propagation. This means that the decision tree decreases (and so the number of selections) according to:

$$\sum_{i=1}^n 2^i / 2 = 2^n - 1 \quad (11)$$

MPR reduces the number of operations for the assignment and propagation in comparison with *CRP* by 50%.

5. CRP vs. MPR

MPR can be used in the design of new algorithms or improve algorithms *CRP* based; but in the case of algorithms *CRP* based, their efficiency it depends on the programming and the philosophy of each one.

```

CRP_Solver ( $\phi$  set of clauses,  $\beta$  set of variables)
  if (exist Unit Clause) then propagate Unit Clauses;
   $\phi[x_k]$ ;
  if ( $\phi = \square$ ) then return unsatisfiable;
  if ( $\beta = \emptyset$ ) then
    return satisfiable;
  Else
     $x_k = (\text{Select a variable in } \beta)$ ;
    if (CRP_Solver ( $(\phi \wedge x_k), (\beta - \{x_k\})$ ) = satisfiable) then
      return satisfiable;
    return CRP_Solver ( $(\phi \wedge \sim x_k), (\beta - \{x_k\})$ );
  Endelse
EndCRP_Solver

```

Fig. 4: CRP_Solver Algorithm.

Some modern algorithms use clauses for the detection and maintenance conflicts. For this reason, to prove to *MRP* against *CRP* basic algorithms were implemented: *CRP_Solver* which uses *CRP* and *UCR* (Fig. 4) and *MRP_Solver* which uses *MRP* and *UCR* (Fig. 5).

Fig. 5: MPR_Solver Algorithm.

```

MRP_Solver ( $\phi$  List of clauses,  $C$  clause,  $\beta$  set of variables)
  if (exist Unit Clause) then propagate Unit Clauses;
  if ( $C > \text{Last Clause}$ ) then return satisfiable;
  if ( $C = \text{Satisfiable}$ ) then
    MRP_Solver ( $(\phi - C)$ , Next clause,  $\beta$ );
  if ( $C = \text{Empty clause}$ ) then return unsatisfiable;
   $x_1 = (\text{first non assigned variable in } C)$ ;
  if (MRP_Solver ( $(\phi[x_1] - C)$ , Next clause,  $(\beta - \{x_1\})$ ) =
    satisfiable) then
    return satisfiable;
   $x_2 = (\text{second non assigned variable in } C)$ ;
  if (exist  $x_2$ )
    if (MRP_Solver ( $(\phi[\sim x_1, x_2] - C)$ , Next clause,
       $(\beta - \{x_1, x_2\})$ ) = satisfiable) then
      return satisfiable;
   $x_3 = (\text{non assigned variable in } C)$ ;
  if (exist  $x_3$ )
    if (MRP_Solver ( $(\phi[\sim x_1, \sim x_2, x_3] - C)$ ,
      Next clause,  $(\beta - \{x_1, x_2, x_3\})$ ) = satisfiable) then
      return satisfiable;
  Endif
  return unsatisfiable;
EndMRP_Solver

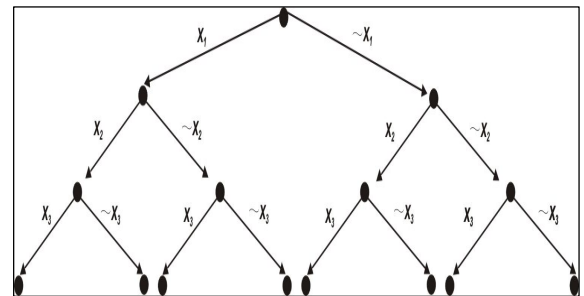
```

To show the *MPR* performance let us consider an example with the following instance ϕ formed by three variables (x_1, x_2, x_3) and eight clauses:

$$\phi = \{(x_1 \vee x_2 \vee x_3), (\sim x_1 \vee x_2 \vee x_3), (\sim x_1 \vee \sim x_2 \vee \sim x_3), (\sim x_1 \vee \sim x_2 \vee x_3), (x_1 \vee \sim x_2 \vee \sim x_3), (x_1 \vee x_2 \vee \sim x_3), (x_1 \vee \sim x_2 \vee x_3), (\sim x_1 \vee x_2 \vee \sim x_3)\} \quad (12)$$

The decision tree formed by *CRP* is shown in the Fig. 6 while the decision tree formed by *MPR* is shown in the Fig. 7. It can be observed that with *MRP* the number of operations is smaller than with *CRP*.

Fig. 6: Assignments for the test instance using *CRP*.



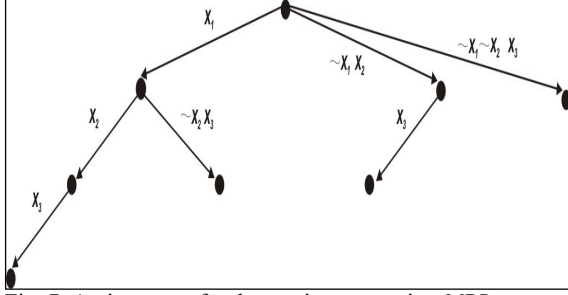


Fig. 7: Assignments for the test instance using *MPR*.

6. CRP vs. MPR

In order to test *MPR_Solver* and to compare it with *CPR_Solver* some instances of SATLIB [15] were considered. The *SAT* instances were separated into two groups: satisfiable (*uf*) and unsatisfiable (*uuf*) and all of these were comprised of 50 variables and 218 clauses.

In Table 1 and Table 2, the comparative results of *MPR_Solver* and *CPR_Solver* are shown. It can be seen that *MPR* is always more efficient than *CPR*.

The obtained results are as follows:

- The results of the *uf* instances are shown in Fig. 8. It can be seen that *MPR* reduces the solution time by 30% with respect to *CPR* (Table 1).
- The results of the *uuf* instances evaluated with *MPR* and *CRP* can be seen in Fig. 9. In these instances, the solution time obtained with *MPR* was decreased about 66% with respect to *CRP* (Table 1).

In general, results show that the *MPR* reduce the solution time of satisfiable and unsatisfiable instances by 57% with respect to *CPR* (Table 2).

Time	MPR <i>uf</i>	CRP <i>uf</i>	MPR <i>uuf</i>	CRP <i>uuf</i>
Average	1,206.89	4,508.50	1,613.52	1,675.70
Medium	989.22	2,973.39	706.39	1,014.87
Maximum	5,535.13	13,294.37	9,931.29	11,836.25
Minimum	231.00	364.86	4.72	35.78

Table 1: Results of instances *uf* and *uuf* using *MPR* and *CRP*.

Time	MPR	CRP
Average	1,410.20	3,092.10
Medium	847.81	1,994.13

Maximum	9,931.29	13,294.37
Minimum	4.72	35.78
	-57.48%	100.00%

Table 2: *MPR* versus *CRP*.

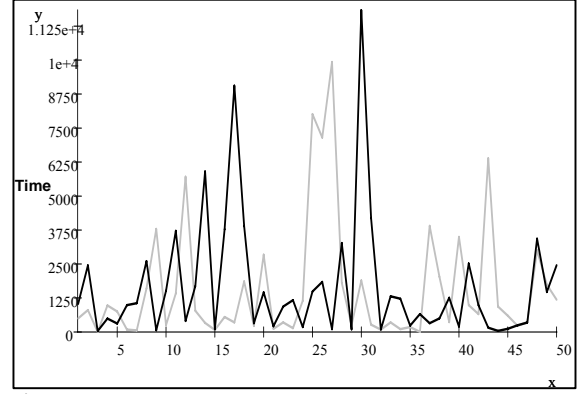


Fig. 8: Results instances *uf* (time): *MPR* (Gray) vs. *CRP* (Black)

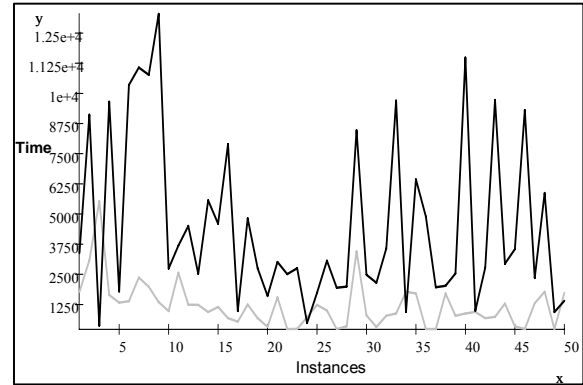


Fig. 9: Results instances *uuf* (time): *MPR* (Gray) vs. *CRP* (Black)

7. Conclusions

In this paper, a new rule named the Multiple Partition Rule (*MPR*) is presented. After conducting a decision tree analysis, it was demonstrated that *MPR* is at least fifty percent more efficient than *CPR* because the maximum number of visited arcs with *MPR* is 2^n in lieu of 2^{n+1} with *CPR*. This theoretical result was tested through experiments using *CRP_Solver* and *MPR_Solver*. The fact is that *CPR* is used in algorithms based on the *DPLL* rules like *zchaff* [10] and *GRASP*[11], therefore an improvement to *CPR* could redound to an improvement of the algorithms

based on her. But even, *MPR* can be used in the creation of a new type of algorithms.

References

- [1] D. Ding-Zhu, G. Jun and P. Panos, Satisfiability problem: Theory and applications, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, 1997.
- [2] S. A. Cook, The complexity of theorem proving procedures, In: *Proceedings of the third Annual ACM symposium on the Theory of Computing*, ACM, 151-158, 1971.
- [3] R. M. Karp, Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, Plenum Press, 85-103, 1972.
- [4] N. Creignou, The class of problems those are linearly equivalent to satisfiability or a uniform method for proving np-completeness. *Lecture Notes in Computer Science* 702, 115-133, 1993.
- [5] S. A. Cook and D. G. Mitchell, Finding Hard Instances of the Satisfiability Problem: A Survey. In Du, Gu, Pardalos, eds.: *Satisfiability Problem: Theory and Applications*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1-17, 1997.
- [6] Philippe Chatalic and Laurent Simon: Davis and Putnam 40 years later: a first experimentation. Technical report, LRI, Orsay, France. citeseer.ist.psu.edu/chatalic00davis.html, 2000.
- [7] G. Logemann, M. Davis and D. Loveland, A Machine Program for Theorem Proving. *Communications of the ACM*, 394-397, 1962.
- [8] M. Davis and H. Putnam, A computing procedure for quantification theory. *J. ACM* 7 201-215, 1960.
- [9] C.L. Chang, R.C. Lee and R.C.T. Lee, Symbolic Logic and Mechanical Theorem Proving. Academic Press, Inc., Orlando, FL, USA, pp. 331. 1997.
- [10] M. Moskewicz, C. Madigan, Y. Zhao, Zhang and Malik: Chaff: Engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference (DAC) Las Vegas*, 7, 2001.
- [11] J.P. Marques-Silva and K.A. Sakallah, GRASP: A Search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506-521, 1999.
- [12] S. Even, A. Itai and A. Shamir, On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.* 5, 691-703, 1976.
- [13] B. Aspvall, M.F. Plass and R.E. Tarjan, A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.* 8 121-123, 1979.
- [14] R. Zabih and McAllester, A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of National Conference on Artificial Intelligence*, 155-160, 1988.
- [15] H. Hoos and T. Stutzle, SATLIB: An Online Resource for Research on SAT. In *SAT 2000*, IOS Press, 283-292, 2000.