Resolution of structured SAT problems with *Score(FD/B)*[1]

Jacqueline CHABRIER, Jean-Michel RICHER, Jean-Jacques CHABRIER
C.R.I.D
Université de Bourgogne
B.P. 138
21004 Dijon Cedex, FRANCE
{jc,richer,chabrier}@crid.u-bourgogne.fr

Abstract

Many satisfiability problems called *structured SAT problems* (like Ramsey, PigeonHole, and so on) may be very shortly defined with *Score(FD/B)*, a boolean constraint programming language design at the Computer Research Center of Dijon (CRID). In addition to the expressiveness, design features of the language (cardinality constraints) are also used to detect some properties (symmetries) of the problems at the compilation of the source program. The detected symmetries (if any) are used by the extended resolution method to prune efficiently the search space. In this paper, we present an original approach which exploits the symmetry properties, statically detected, to improve the systematic local search method supported by *Score(FD/B)*. We also describe how to make use of these properties on the applications Ramsey and PigeonHole. Then we report some experimental results.

The system *Score(FD/B)* has been developped in C on a SUN SPARC 10.

Keywords

SAT, boolean constraint language, symmetries, local search, systematic method.

1 Introduction

Propositional satisfiability (SAT) is the problem of deciding if there is an assignment for the variables in a propositional formula that makes the formula true. SAT is of considerable practical interest as many AI tasks can be encoded as SAT problems (eg. theorem proving [19], constraint satisfaction [1], interpretation [16], planning [12]). Unfortunately, unless $P = NP$, SAT is intractable in the worst case, as it is a NP-hard problem. In recent years, there are, however, a number of theoretical and experimental results which produce good performance for certain classes of SAT problems (GSAT [17], GENET [8], CSAT [10]). Other works in constraint programming language design (PROLOG III [7], CHIP [20], ILOG SOLVER [11]) propose several language features

[1]*Score(FD/B)* is the acronym of System of COnstraints and associated REpairs methods for solving Constraint Satisfaction Problems on Finite Domains (FD) restricted to Booleans (B). This research is partially supported by the PRC-IA BAHIA.

for describing concisely constraint satisfaction problems on boolean and numerical domains [22], and help users to solve their problems with constraint satisfaction techniques.

Certain kinds of SAT problems that we call *structured SAT problems*, because they can be described in a high level language with small sets of boolean constraints (for example cardinality constraints), are difficult. Often, they can not be solved with the classical methods of resolution if they involve more than a certain number of variables. There are, however, theoretical and experimental studies dealing with symmetry properties [3] and cardinality formulas in propositional calculus [4], which show good results for several known problems such as Schur'lemma, the eight queens, and Ramsey. The most interesting result is the proof of unsatisfiability of the Ramsey problem for 17 vertices and 3 colors. The symmetries introduced in different automated deduction algorithms like SL-Resolution [13], Davis & Putnam procedure [9] and semantic evaluation [15] are detected and used dynamically at each step of the resolution.

In this paper, we present a new method which uses symmetry properties detected at the compilation of structured SAT programs described with the *Score(FD/B)* language. Indeed, this method is an extended version of the complete local-based search method presented in [6]. In section 2, we give a brief overview of *Score(FD/B)*: some characteristic features of the boolean constraint language and the associated resolution method. In section 3, we introduce the necessary terminology with definitions on symmetries in the propositional calculus. In section 4, we propose a method which uses symmetries detected statically to compute only the basic models (non-symmetrical models). In section 5, we describe how to make use of these properties on the applications Ramsey and PigeonHole and then we report some experimental results with and without symmetries. We end in section 6 with some conclusions.

2 A brief overview of *Score(FD/B)*

Score is a general framework of constraint system developed at the Computer Research Center of Dijon by Jacqueline Chabrier to solve constraint satisfaction problems on finite domains. The *Score* instance on booleans (0/1) domains, called *Score(FD/B)*, integrates a compiler of the boolean constraint language and a dedicated complete local-based search method.

2.1 The *Score* language

For a SAT problem defined by a formula in a conjunctive normal form (CNF) i.e. a conjunction of clauses where a clause is a disjunction of literals and a literal a negated or un-negated variable, the translation in *Score(FD/B)* is direct. Indeed, a clause is considered as a boolean constraint, i.e. a relationship between the boolean variables of the clause; and a formula in CNF as a set of boolean constraints. Note that these basic constraints of the language are useful to describe random SAT problems [6]. Now, many problems present some structures we want to take into account at the language level. For this purpose, *Score(FD/B)* supports new language design features: parameterized modules, arrays of boolean variables with iteration and if statements, and a powerful built-in boolean constraint: the cardinality constraint [21]. Some complete programs are given in the following, so we only introduce here the new cardinality constraint: $\#(p, q, l_1, l_2, \ldots, l_n)$, for $p, q, n \in N, 0 \leq p \leq q \leq n$, which means that at least p and at most q literals among $l_1, l_2, \ldots, l_n$ must be true.

Using this cardinality constraint, many problems have a shorter description than with basic constraints defined before. Morever, each cardinality constraint is equivalent to a system of basic constraints.

2.2 The *Score* method

The *Score* method presented in [6] combines advantages of local search and systematic techniques. We will assume that the reader is already familiar with these approaches [18].

Score can be considered as a local search method but with a memory mechanism. The method uses an initial variables configuration and searchs the next one in the neighborhood, until the solution is found. The added memory mechanism avoids to test twice the same path, and also makes sure, if necessary, that all the search tree will be explored. Therefore, this approach provides a complete method and allows to find all the solutions.

Score can also be considered as a systematic method since, during resolution, some instanciated variables produce a partial solution. But, in our case, all the variables are instanciated and therefore the current configuration can be used in the choice heuristics.

The input of the *Score* procedure is a data structure produced by the compiler. This structure includes a representation in the domain (0/1) of the set V of the boolean variables declared in the source program and a representation of the set C of all the source constraints unfolded at compilation. *Score* performs a local-based search to satisfy the set C of constraints. One important feature of *Score*, however, is the completeness of the method discussed in [6].

The procedure starts with an initial configuration, i.e. an assignment of values (0/1) for all the variables in V partitioned in three disjoint subsets VM, VC and VNC. The subset VM memorizes *marked* variables, i.e. variables considered without conflict in this subset (partial solution). VC is the subset of variables in conflict and VNC the subset of the other variables. A procedure called *repair*, using the neighborhood of the current configuration, marks a variable with an assignment that increases the set VM of the marked variables. If, for a chosen variable and its associated value, an inconsistency occurs, *repair* chooses another value. When the two values for this variable have failed, *repair* backtracks. Such repairs are repeated until either a satisfying assignment is found or the procedure fails.

3 Symmetries in propositional calculus

The idea of using symmetries in propositional calculus to obtain short proofs was first introduced by Krishnamurty [14]. He uses the fact that a set of constraints remains invariant by a permutation of the literals, improving so the performances by enabling entire subsets of the search space.

First of all, let us give some definitions and theorems on symmetries that are useful in the following [2].

Let $\mathcal{C}$ be a set of constraints expressed on a set $\mathcal{L}$ of literals.

Definition 1 - Interpretation - *An interpretation of a set L of literals, $L \subseteq \mathcal{L}$, is an application I of $L \rightarrow \{0,1\}$ such that $\forall l \in L$, $I(\neg l) = \neg I(l)$.*

Definition 2 - Model for $\mathcal{C}$ - *An interpretation I of $\mathcal{L}$ is a model for $\mathcal{C}$ if, with this interpretation, c is true $\forall c \in \mathcal{C}$.*

Definition 3 - Model for (L, I) - *A pair (L, I), where $L \subseteq \mathcal{L}$ and I is an interpretation of L, has a model if I can be expanded to $\mathcal{L}$ to obtain a model for $\mathcal{C}$.*

Definition 4 - Symmetry - *A permutation $\sigma : \mathcal{L} \rightarrow \mathcal{L}$ is called a symmetry of $\mathcal{C}$ if it satisfies the following properties:*

1. $\forall l \in \mathcal{L}, \sigma(\neg \updownarrow) = \neg\sigma(\updownarrow)$
2. $\sigma(\mathcal{C}) = \mathcal{C}$

Definition 5 - Symmetrical literals - *Two literals l and l' of $\mathcal{L}$ are symmetrical in $\mathcal{C}$ ($l \sim l'$) if there exists a symmetry σ of $\mathcal{C}$ such that $\sigma(l) = l'$.*

Theorem 1 - *If $l \sim l'$, (l, I) has a model iff (l', I) has a model.*

4 Symmetries with *Score(FD/B)*

4.1 Principle

Due to features of the language *Score(FD/B)* such as cardinality constraints, arrays and foreach structures, we can detect statically (at the compilation time) some symmetries.

The usual application of symmetries is very simple. Suppose we have a symmetry property between two literals l and l' of $\mathcal{L}$, noted $P(l, l')$, then if $(\{l, l'\}, I)$ has no model, we can deduce that $(\{l, l'\}, I')$ has no model for some other interpretations I'. This is the direct consequence of Theorem 1. This result permits significant cuts in the search tree.

In this section, we propose to generalize the symmetry property P to a set L of literals of $\mathcal{C}$ in order to have a similar result for the interpretations and an attractive pruning of the search space.

In the following, we will consider $n * m$ literals and the matrix noted $M[I]$ which represents an interpretation I of these literals.

Definition 6 - $M \overset{*}{\Longrightarrow} M'$ - *If M and M' are two matrices, we note $M \overset{*}{\Longrightarrow} M'$ if M' is obtained from M by swapping as many times as wanted two rows or two columns.*

Definition 7 - [a], [b] and [c] properties - *If $L = \{l_{ij}\}$ is a set of $n * m$ literals of $\mathcal{C}$, we can define the following properties:*

[a] $\exists C_n^2$ symmetries $\sigma_{pq}^{(1)}$, for $p, q = 1, \ldots, n$ $p < q$, such that

$$l_{pj} = \sigma_{pq}^{(1)}(l_{qj})\ \forall j = 1, \ldots, m$$
$$l_{qj} = \sigma_{pq}^{(1)}(l_{pj})\ \forall j = 1, \ldots, m$$
$$l_{\alpha j} = \sigma_{pq}^{(1)}(l_{\alpha j})\ \forall \alpha = 1, \ldots, n \text{ s.t. } \alpha \neq p, q\ \forall j = 1, \ldots, m$$

[b] $\exists C_m^2$ symmetries $\sigma_{pq}^{(2)}$, for $p, q = 1, \ldots, m$ $p < q$, such that

$$l_{ip} = \sigma_{pq}^{(2)}(l_{iq})\ \forall i = 1, \ldots, n$$
$$l_{iq} = \sigma_{pq}^{(2)}(l_{ip})\ \forall i = 1, \ldots, n$$
$$l_{i\alpha} = \sigma_{pq}^{(2)}(l_{i\alpha})\ \forall \alpha = 1, \ldots, m \text{ s.t. } \alpha \neq p, q\ \forall i = 1, \ldots, n$$

[c] $\exists$ m cardinality constraints, for $j = 1, \ldots, m$

$$\#(1, 1, l_{1j}, l_{2j}, \ldots, l_{nj}).$$

Theorem 2 - *Let $L = \{l_{ij}\}$ be a set of $n * m$ literals of $\mathcal{C}$ which satisfy the properties [a] and [b]. Then if (L, I) with $M[I]$ has no model, then (L, I') with $M'[I']$ has no model if $M \overset{*}{\Longrightarrow} M'$.*

Proof - The existing symmetry $\sigma_{pq}^{(1)}$ proves that if (L, I) with $M[I]$ has no model, then (L, I') with $M'[I']$ has no model if M' is obtained from M with the swap of the two rows p and q.

Likewise, the existing symmetry $\sigma_{pq}^{(2)}$ proves that if (L, I) with $M[I]$ has no model, then (L, I') with $M'[I']$ has no model if M' is obtained from M with the swap of the two columns p and q.

The set of the existing symmetries $\sigma_{pq}^{(1)}$ and $\sigma_{pq}^{(2)}$ proves the result.

In order to apply this theorem, we must find the symmetries; but also, with the given matrix M and M', verify if $M \overset{*}{\Longrightarrow} M'$.

Theorem 3 - *Let $M(n, m)$ and $M'(n, m)$ be two matrices such that $\forall i, 1 \leq i \leq n$, $r_i = \sum_{j=1}^{m} M_{ij}$ and $r'_i = \sum_{j=1}^{m} M'_{ij}$. If $M \overset{*}{\Longrightarrow} M'$, then $(r'_1, \ldots, r'_n)$ is a permutation of $(r_1, \ldots, r_n)$.*

Proof - When we swap two columns of M, we obtain a matrix M' with $r'_i = r_i$ $\forall i = 1, \ldots, n$. When we swap two rows of M, we obtain a matrix M' where $(r'_1, \ldots, r'_n)$ is a permutation of $(r_1, \ldots, r_n)$. We have the wanted result.

We have a necessary condition for $M \overset{*}{\Longrightarrow} M'$. But it is not a sufficient condition. If $c_j = \sum_{i=1}^{n} M_{ij}$ $\forall j = 1, \ldots, m$ and $c'_j = \sum_{i=1}^{n} M'_{ij}$ $\forall j = 1, \ldots, m$ and if we add the condition that $(c'_1, \ldots, c'_m)$ is a permutation of $(c_1, \ldots, c_m)$, we still do not obtain a sufficient condition. A counter-example is given below:

$$M = \begin{matrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{matrix} \qquad M' = \begin{matrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{matrix}$$

Theorem 4 - *Let $M(n, m)$ and $M'(n, m)$ be two matrices such that $r_i = \sum_{j=1}^{m} M_{ij}$, $c_j = \sum_{i=1}^{n} M_{ij}$, $r'_i = \sum_{j=1}^{m} M'_{ij}$ and $c'_j = \sum_{i=1}^{n} M'_{ij}$. If $(r'_1, \ldots, r'_n)$ is a permutation of $(r_1, \ldots, r_n)$ and if $c_j = c'_j = 1$ $\forall j = 1, \ldots, m$, then $M \overset{*}{\Longrightarrow} M'$.*

Proof - We show that we can, by successive permutations of two rows or two columns transform each of the two matrices M and M' to a matrix M'' ($r''_i = \sum_{j=1}^{m} M''_{ij}$ and $c''_j = \sum_{i=1}^{n} M''_{ij}$) such that $c''_j = 1\ \forall j = 1, \dots, m$, $(r''_1, \dots, r''_n)$ is a permutation of $(r_1, \dots, r_n)$ and of $(r'_1, \dots, r'_n)$, $r''_1 \geq \dots \geq r''_n$ and $M''_{ij} = 1$ if $r''_1 + \dots + r''_{j-1} < j \leq r''_1 + \dots + r''_j$, and 0 else.

Theorem 5 - Main result - *Let $\{l_{ij}\}$ a set of $n * m$ literals with the properties [a], [b] and [c]. If (L, I) with $M[I]$ has no model, then (L, I') with $M'[I']$ such that $(r'_1, \dots, r'_n)$ is a permutation of $(r_1, \dots, r_n)$ has no model.*

Proof - This theorem results from Theorems 2 and 4.

4.2 Symmetry use

At the compilation of the source program, we try to find $n * m$ literals which satisfy the properties [a], [b] and [c]. Note that our method does not search all the symmetries of the problem.

First at all, we begin by searching if we have the property [c] for some cardinality constraints, because it is the condition to obtain the sets of literals $\{l_{1j}, \dots, l_{nj}\}$. We verify if there exist symmetries satisfying the [a] property on all or a subset of these literals. If it is the case, we then verify if we have the other symmetries with the [b] property.

When we have found the $n * m$ literals $\{l_{ij}\}$ satisfying [a], [b] and [c], we say that if $(\{l_{ij}\}, I)$, with the values $\{r_1, \dots, r_n\}$ for $M[I]$, has no model, then $(\{l_{ij}\}, I')$,where $\{r'_1, \dots, r'_n\}$ for $M'[I']$ is a permutation of $\{r_1, \dots, r_n\}$, has no model.

Now, we consider an extended version of resolution method *Score* to take into account the symmetry properties. The procedure starts with an initial configuration and then tries to mark the variables associated with the literals $\{l_{ij}\}$ found during the compilation phase. When these variables with their interpretation have no model, then we can eliminate an important number of other interpretations for these variables for which we know they have no model. Therefore, the procedure will not have to visit the corresponding branches in the proof tree.

Note that this *Score* procedure makes important cuts in the search tree for satisfiable problems as well as for unsatisfiable ones. See the Ramsey application.

5 Applications

5.1 Ramsey

The Ramsey problem (3, 3, 3; 2) consists in coloring the edges of a full graph of N vertices with three colors such that there is no monochromatic triangle in the graph.

For $N = 3$ to 16, the problem is satisfiable. For $N \geq 17$, the problem is unsatisfiable, but this property is not provable in an acceptable time with the basic version of *Score*.

```
proc Ramsey(N)
{
    int    I,J,K;
    array  red[N][N], green [N][N], blue [N][N];
    /* One edge has one and only one color */
    foreach I in 1..N-1
        foreach J in I+1..N
            #(1,1, red[I][J], green[I][J], blue[I][J] );
    /* There is no monochromatic triangle */
    foreach I in 1..N-2
        foreach J in I+1..N-1
            foreach K in J+1..N
            {
                #(0,2, red[I][J], red[I][K], red[J][K] );
                #(0,2, green[I][J], green[I][K], green[J][K] );
                #(0,2, blue[I][J], blue[I][K], blue[J][K] );
            }
}
```

Figure 1: Ramsey problem

If we want to use the main theorem, at the compilation phase, we find the cardinality constraints $\#(1,1,red[I][J],green[I][J],blue[I][J]) \forall I = 1,\ldots,N-1$ and $J = I+1,\ldots,N$. Therefore we have the [c] property and $n = 3$. Then, we find $C_3^2 = 3$ symmetries $\sigma_{rg}^{(1)}$, $\sigma_{rb}^{(1)}$ and $\sigma_{gb}^{(1)}$ corresponding to the three symmetries between the colors red and green, red and blue, green and blue. After that, we search for symmetries $\sigma_{pq}^{(2)}$ between only literals $red[I][J]$, only literals $green[I][J]$ and only literals $blue[I][J]$.

We find $(N-1)(N-2) = C_{N-1}^2$ symmetries on the [1][2],[1][3],…,[1][N] suffixes for the three sets of literals.

So we have $n = 3, m = N-1$,

$$L = \begin{bmatrix} red[1][2] & red[1][3] & \ldots & red[1][N] \\ green[1][2] & green[1][3] & \ldots & green[1][N] \\ blue[1][2] & blue[1][3] & \ldots & blue[1][N] \end{bmatrix}$$

For $N = 17$, we only test 30 markings for 48 variables. The *Score* method finds very quickly that there is no model.

For $N = 16$, this method improves the search time which decreases from 3 minutes 45 seconds to 1 second; many branches are pruned before finding the model.

5.2 PigeonHole

The PigeonHole problem consists in finding an allocation of N pigeons in P dovecotes, where each dovecote can contain 1 pigeon maximum.

We know that for $N \leq P$, the problem is satisfiable and a model can be found in a linear time; and for $N > P$, the problem is unsatisfiable and we find it in an exponential time.

---

proc PigeonHole(N, P)
{
  int I;
  array pig[N][P];
  /* One pigeon is in one and only one dovecote*/
  foreach I **in** 1..N
    #(1, 1, pig[I][*]);
  /* In a dovecote, there is 1 pigeon maximum */
  foreach I **in** 1..P
    #(0, 1, pig[*][I]);
}

Figure 2: PigeonHole problem

We find N cardinality constraints: $\#(1, 1, pig[I][1], pig[I][2], \ldots, pig[I][P]])$ $\forall I = 1, \ldots, N$, and we have [c] and $n = P$. Then we find C_P^2 symmetries $\sigma_{pq}^{(1)}$ between $pig[I][p]$ and $pig[I][q]$ satisfying [a], and C_N^2 symmetries $\sigma_{pq}^{(2)}$ between $pig[p][I]$ and $pig[q][I]$ satisfying [b].

So we have $n = P, m = N$, and

$$L = \begin{bmatrix} pig[1][1] & pig[2][1] & \ldots & pig[N][1] \\ pig[1][2] & pig[2][2] & \ldots & pig[N][2] \\ \ldots & & & \\ pig[1][P] & pig[2][P] & \ldots & pig[N][P] \end{bmatrix}$$

In conclusion, for $N > P$, if the allocation of P particular pigeons does not give a model, then the allocation of any others pigeons neither will give any model. Therefore, the procedure prunes a lot of the search space. On the other hand, for $N \leq P$, the improvement is not so remarkable because the solutions are immediately found by the procedure.

5.3 Results

In Table 1, we present results for the two applications Ramsey and PigeonHole on a SUN SPARC Station 10. We give in the first column the name of the problem, then the satisfiability (Y or N), the number of variables, the number of cardinality constraints and in the two last columns the CPU time without and with symmetries.

6 Conclusion

In this paper, we have shown that a powerful constraint language improves the expressiveness but also facilitates the detection of some properties of the prob-

Problem	Sat	Vars	Constraints	Time without symmetries	Time with symmetries
PigeonHole 10,10	Y	100	20	1 s	1 s
PigeonHole 10,9	N	90	19	45 s	1 s
PigeonHole 11,10	N	110	21	5 m 40 s	10 s
PigeonHole 30,30	Y	900	60	1 s	1 s
PigeonHole 30,29	N	870	59	∞	55 s
Ramsey 14	Y	273	1183	1 s	1 s
Ramsey 15	Y	315	1470	48 s	1 s
Ramsey 16	Y	360	1800	3 m 45 s	1 s
Ramsey 17	N	408	2176	∞	1 s

Table 1: Results with *Score(FD/B)*

lems. The detected symmetries (if any) may be used by an extended resolution method to prune efficiently the search space. We believe that symmetries are especially suitable for dealing with many structured problems.

To conclude, symmetries through the method presented in this paper, make it possible to improve any other resolution, if the use of symmetries help us to prune the search space and if we don't lose a lot of time in detecting them. That is the case here thanks to the language.

Acknowledgements The authors wish to thank the anonymous referees for their valuable comments on an early draft of this paper.

References

[1] Projet BAHIA. *Etude comparative de trois formalismes en calcul propositionnel : projet BAHIA, (Booléens, heuristiques et algorithmes pour l'I.A.)* 5èmes Journées Nationales PRC-GDR I.A., (C.N.R.S., M.R.T.), Nancy, 1995.

[2] Belaid Benhamou, Lakhdar Sais. *Formules de cardinalité et symétries en calcul propositionnel*, Master's thesis, GIA, Marseille, 1993.

[3] B. Benhamou, L. Sais and P. Siegel. *Dealing with symetry in propositional calculus*, AAAI 92, San Jose, 1992.

[4] B. Benhamou, L. Sais and P. Siegel. *Dealing with cardinality formulas in propositional calculus*, AAAI 92, San Jose, 1992.

[5] J. Chabrier, J. J. Chabrier, F. Trousset. *Résolution efficace d'un problème de satisfaction de contraintes : le million de reines*, 11èmes Journées Internationales d'Intelligence Artificielle, Avignon, Mai 1991.

[6] J. Chabrier, V. Juliard, J. J. Chabrier. $\mathcal{S}$core*(FD/B): An efficient complete local-based search method for satisfiability problems*, CP95, Workshop on Studying And Solving Really Hard Problems, Cassis, September 1995.

[7] A. Colmerauer. *An introduction to PROLOG III*, CACM 33(7), p 69-90, 1990.

[8] A. J. Davenport, E. P. K. Tsang, C. J. Wang, K. Zhu. *GENET: a connectionnist architecture for solving constraint satisfaction problems by iterative improvement*, In Proc. 12th National Conf. on Artificial Intelligence, Vol 1, p 325-330, 1994.

[9] M. Davis and H. Putnam. *A computing procedure for quantification theory*, JACM 7, p 201-215, 1960.

[10] O. Dubois, P. André, Y. Boufkhad, J. Carlier. *SAT vs UNSAT*, DIMACS Challenge on Satisfiability Testing, 1994.

[11] ILOG SOLVER. *Reference Manual*, Version 3.0, 1995.

[12] H. A. Kautz, B. Selman. *Planning as satisfiability*, In Proc. of the 10th ECAI, p 359-363, 1992.

[13] R. Kowalski, D. Kuehner. *Linear resolution with selection function*, Artificial Intelligence (2), p 227-260, 1971.

[14] B. Krishnamurthy. *Short Proofs for Tricky Formulas*, Acta Informatica, vol 44, pages 253–275, 1985.

[15] L.Oxusoff, A. Rauzy. *L'évaluation sémantique en calcul propositionnel*, PhD Thesis, GIA, Marseille, 1989.

[16] R. Reiter, A. Mackworth. *A logical framework for depiction and image interpretation*, Artificial Intelligence, 41(3), p 123-155, 1989.

[17] B. Selman, H.J. Levesque, D.G. Mitchell. *A New Method for Solving Hard Satisfiability Problems*, Proc. of AAAI-92, San Jose, CA, p 440-446, 1992

[18] E. Tsang *Foundations of constraint satisfaction*, Academic Press, 1993.

[19] G. S. Tseitin. *On the complexity of derivation in propositional calculus*, Structures in the constructive mathematics and mathematical Logic, p 115-125, H.A.O. Shsenko, 1968.

[20] P. Van Hentenryck. *Constraint Satisfaction in Logic programming*, Logic programming Series, The MIT Press, Cambridge, MA, 1989.

[21] P. Van Hentenryck and Y. Deville. *The Cardinality Operator : A New Logical Connective for Constraint Logic Programming*, In Proc. of the 8th ICLP, Paris, 1991, p 745-759, The MIT Press, Cambridge, Mass. 1991.

[22] P. Van Hentenryck, H. Simonis, M. Dincbas. *Constraint satisfaction using constraint logic programming*, Artificial Intelligence 58, p 113-159, 1992.