ISSN 1870-4069

On the Use of CSP Semantic Information in SAT Models

Claudia Vasconcellos-Gaete, Vincent Barichard, Frédéric Lardeux

Université d'Angers, Université Bretagne Loire,
Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA),
Angers, France
{claudia.vasconcellos, vincent.barichard,
frederic.lardeux}@univ-angers.fr

Abstract. Constraint Satisfaction Problems (CSP) and Propositional Satisfiability Problems (SAT) are two paradigms intended to deal with constraint-based problems. In CSP modeling, it results natural to differentiate between *decision* and *auxiliary* variables. In SAT, instances do not contain any information about the nature of variables; solvers use the Variable Selection heuristic to determine the next decision to make. This article studies the effect of transfer semantic information from a CSP model to its corresponding SAT instance, in order to guide the branching only to variables directly related to the CSP model. The results obtained suggest that this modification can speed up the resolution for some instances.

Keywords: CSP, SAT, decision variables.

1 Introduction

Constraint Satisfaction Problems (CSP) and Propositional Satisfiability Problems (SAT) are two paradigms intended to deal with constraint-based problems.

A CSP problem ($\mathcal{P}$) is defined as a triple $\mathcal{P} = \langle X, D, C\rangle$. This triple contains a set of variables $X = \{x_1, x_2 \ldots, x_n\}$, a set of finite domains $D = \{d_1, d_2 \ldots, d_n\}$ and a set of constraints $C = \{C_1, C_2 \ldots, C_m\}$. A constraint C_j is a relation between the domains $c \subseteq D_1 \times \ldots \times D_n$.

Usually, a CSP model has decision and auxiliary variables. Decision variables represent any value that decision maker needs to determine and they are used by the solver as decision points. Auxiliary variables could be introduced to support the modeling; they are not decision points, but they are invoked during propagation.

A SAT problem ($\mathcal{S}$) is defined as a tuple $\langle X, L, \Phi\rangle$. This triple contains a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$, a set of literals $L = \{l_{1,1}, l_{1,2}, \ldots, l_{2n}\}$ and a Boolean formula in Conjunctive Normal Form (CNF) $\Phi : \{0,1\}^n \to \{0,1\}$. A literal represents a variable (x_i) or its negation ($\neg x_i$). The problem is *satisfiable* only if there exists an assignment of truth values for X in which the formula Φ is true; otherwise, the problem is *unsatisfiable*.

Modeling a problem directly in SAT is a complex task, so there are almost no problems modeled originally as a SAT formula. Also, formats to describe SAT CNF instances (i.e.: DIMACS CNF) are limited describing models; for example, they do not differentiate between decision or auxiliary variables. Typically in a SAT (CNF) instance all variables are equally considered (in their semantics) to be picked as the next variable to branch.

In this article, we study if by adding CSP semantic information to a SAT instance the resolution can improve, basing the analysis in restricting the set of variables with the potential to be picked by the SAT variable selection heuristic. Also, we propose a graph-based tool able to receive a CSP model, produce its corresponding SAT model and trace (in a single graph) all the transformations occurred in the path from CSP to SAT. The results show that focusing branching only in variables directly related to the CSP model, it is possible to reduce the number of decisions made by solver.

This article is structured as follows: Section 2 introduces the framework proposed to address CSP and SAT modeling languages, Section 3 presents a discussion about the branching heuristics in CSP and SAT solvers and how information could be kept when going from CSP to SAT. Section 4 presents the Magic Square problem. Finally, the results obtained are given in Section 5 and the upcoming work is discussed at Section 6.

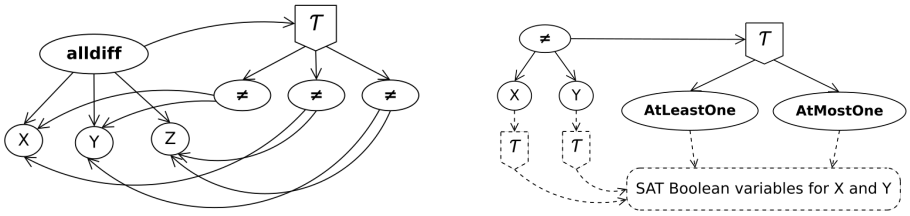
2 Transforming CSP Models into SAT Models

Based on the main elements from CSP (variables, domains and constraints) and SAT (Boolean variables, literals and clauses) we propose a graph able to support both specifications by generalizing them in terms of: *variables*, *relations* and *transformations*. Variables are any symbol representing a value in a finite domain specified by the problem. Relations represents any relationship between one or more variables in the same space (CSP constraint, SAT clause, etc.). Transformations are any function able to produce a set of variables and relations that conserves the original solution (like CSP to SAT encodings).

2.1 The Graph-based Model

We propose an *acyclic labeled graph* $G = (\mathcal{N}, \mathcal{E})$ where the set $\mathcal{N}$ of nodes is composed of three non overlapped sets $\mathcal{N} = \mathcal{V} \bigcup \mathcal{R} \bigcup \mathcal{T}$ with $\mathcal{V} \bigcap \mathcal{R} = \emptyset$, $\mathcal{V} \bigcap \mathcal{T} = \emptyset$ and $\mathcal{R} \bigcap \mathcal{T} = \emptyset$. $\mathcal{V}$ is a subset of nodes labeled with variables names, $\mathcal{R}$ is a subset of nodes labeled with CSP constraints or SAT clauses names, and $\mathcal{T}$ is a subset of nodes labeled with transformation names. The set of edges $\mathcal{E} = \{e_1, \ldots, e_m\}, e_i : n_x \rightarrow n_y \mid n_x, n_y \in \mathcal{N}$ expresses link between two elements. For example, Figure 1 shows the graph representations for transformations `alldiff_to_diff()` and `diff_to_cnf()`.

Using a single graph to maintain the CSP and SAT models allow us to profit from the classical graph operations. For example, a search operation can trace the path followed by a variable or constraint from CSP to SAT and viceversa,



(a) `alldiff_to_diff()` (b) `diff_to_cnf()`

Fig. 1. Graph representation of some CSP/SAT relations and transformations.

determine when a SAT variable was created, identify the relations working with some variables, among others.

Finally, the CSP to SAT encoding chosen produce the SAT model in CNF, facilitating the generation of the corresponding DIMACS CNF output.

2.2 SAT Encoding for Arithmetic Constraints

We extended the idea described in [5] to encode linear arithmetic constraints with digital adders. To be compatible with the machine representation required by them, we use *Log Encoding* [7]. In this encoding, each integer variable v_i is represented with n Boolean variables x_i^k, where $x_i^k = 1$ only if the k-th bit of the domain value assigned to v_i is 1.

To encode a linear constraint $\sum a_i x_i = c$ we consider the schema of a full adder. As this schema only permits to sum two values at the same time, we decompose the linear constraint into a bunch of sums in the form $x + y = z$. Each logical gate correspond to one or more CNF clauses, and the intermediate results (between gates) were modeled with Boolean auxiliary variables.

3 Decision versus Auxiliary Variables

In CSP modeling, a *decision variable* represents values that a decision maker needs to determine. Also, some *auxiliary variables* could be introduced to support the modeling when constraints are difficult to express or, to help the model to propagate better. This difference between variables is used later by the *variable ordering heuristic* to extend nodes in the search tree until all decision variables have been valuated.

Unlike CSP, SAT does not differentiate between decision or auxiliary variables directly in the model. Instead, what some SAT solvers call decision variable is actually any Boolean variable chosen by the branching heuristic during the solving phase. Then, if the branched variables are related (or not) to the decision variables in the problem or if they are only to store intermediate results, it is not concerning to the SAT solver.

In the graph proposed, the semantic information is transferred from CSP to SAT during the encoding of the CSP model; then, the origins of SAT Boolean

variables can be traced to the variable encoding (Log encoding in this case) or as part of the constraint encoding. Moreover, the variable encoding can apply over CSP decision or auxiliary variables (Figure 2).

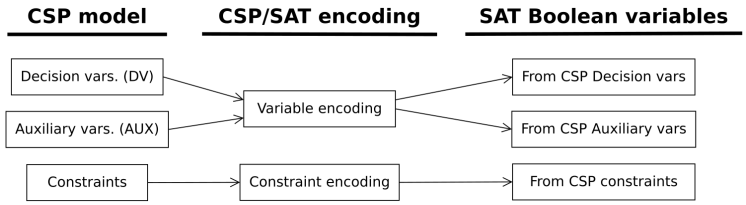


Fig. 2. Origins of SAT Boolean variables.

As our interest is to investigate if by adding CSP semantic information to a SAT instance the resolution can improve, we based the analysis in restricting the set of variables with the potential to be picked by the SAT Variable Selection heuristic. Technically, the graph tags all SAT Boolean variables in the instance as *decision* or *auxiliary* (based on the CSP model given) and then, the DIMACS CNF output generator will sort the variables and add a new parameter to the CNF to indicate the first ID corresponding to the auxiliary variables. The new description in the DIMACS CNF file will be:

```
p cnf nvars nclauses first_auxiliary_var
```

4 The Magic Square

The *Magic Square* (MS) is a mathematical puzzle that consists in finding an assignment of different natural values for a $N \times N$ matrix so that the sum across the rows, columns and diagonals always results in the same number, called the *magic number* [4]. The number N in this puzzle is known as *order*.

In the classical version of the problem, variables are in the domain $[1, N^2]$ and the magic number is calculated as $M = N(N^2 + 1)/2$ (the *open* version of MS uses non-consecutive values in the square, but it is not our interest to consider this case).

The CSP model for Magic Square is defined by the following constraints (Equations 1, 2, 3 and 4):

$$\forall\, i, j \in \{1, N\} \quad \text{alldifferent}(x_{ij}), \tag{1}$$

$$\forall\, i \in \{1, N\} \quad \sum_{j=1}^{N} x_{ij} = M, \tag{2}$$

$$\forall\, j \in \{1, N\} \quad \sum_{i=1}^{N} x_{ij} = M, \tag{3}$$

$$\sum_{i=1}^{N} x_{ii} = \sum_{i=1}^{N} x_{i(N-i+1)} = M, \tag{4}$$

$$x_{ij} \in \{1, N^2\}. \tag{5}$$

5 Experimental Results

All experiments reported in this section use instances of the Magic Square of orders between 4 and 9. They were performed on a machine Intel® Core™ i7-2620M CPU @ 2.70GHz (quad-core, 64 bits) with 8 GB RAM, running Ubuntu 16.04 LTS. Runtimes reported are in seconds, with an upper limit of 2 hours (beyond that time, we will report ∞ values). The solvers used are Gecode (version 5.0.0) [6], and Glucose (version 4.1), a CDCL SAT solver [1].

All instances have been produced with the graph proposed. Variables were encoded in Log encoding and linear constraints are based in the full adder described in subsection 2.2.

5.1 CSP Solving

First, we measure the effort required by a CSP solver to deal with the MS selected instances. For each instance we developed a full CSP model, using Global Constraints and symmetry breaking ("Standard CSP model") and a second model, with no Global constraints and no symmetry breaking ("Decomposed CSP").

The values reported are: runtime (in seconds), the number of propagations and the number of failures occurred during the solving.

Table 1. CSP Results for the Magic Square.

	Instance	Solutions Found	Runtime	Propagations	Failures
Standard CSP	MS4	1	0.014	27152	892
	MS5	1	0.443	2292251	72227
	MS6	1	0.001	1382	27
	MS7	1	2.698	9841603	481301
Decomposed CSP	MS4	1	0.001	2075	14
	MS5	1	0.011	82083	498
	MS6	1	0.847	7866816	47162
	MS7	1	∞	∞	∞

The results in Table 1 show how the difficulty increases when symmetry is allowed in the model (MS7 case), even when the instances have known magic numbers (M) and the domain sizes are limited to the range $[1, N^2]$.

5.2 Structure of CNF Instances Obtained

We describe the structure of the SAT CNF instances produced (in terms of variables and clauses) and observe the changes that a typical SAT minimization step (preprocessing) can do over them.

Structure of Clauses pre/post Minimization Based on Glucose preprocessor (option `-dimacs`), we compare how the structure of the SAT CNF instance changes before/after the minimization (*Raw* versus *Minimized* CNF). Values reported are number of variables, number of clauses and distribution of clauses by arity. The "raw" instances are the CNF obtained using the proposed graph.

Table 2. Structure of SAT CNF instances produced.

	Instance	vars	clauses	CNF clause arity									
				10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Raw CNF	MS4	1773	6829	-	-	-	20	376	-	120	5010	1260	43
	MS5	3674	14008	-	-	24	12	475	-	192	11004	2232	69
	MS6	7484	29528	-	28	28	1638	-	-	280	23842	3612	100
	MS7	12728	49825	16	48	16	1911	-	-	384	42032	5280	138
Minimized CNF	MS4	675	3688	-	-	603	10	178	16	63	2520	278	-
	MS5	1378	7984	-	-	2200	29	158	25	-	5154	398	-
	MS6	3089	17794	-	4744	13	277	36	-	240	12235	249	-
	MS7	5390	31214	-	9096	71	292	49	-	60	21120	526	-

Results in Table 2 show that minimized instances reduced -in average- a 60.2% of variables and a 41.5% of clauses. Regarding binary clauses, they do not represent more than a 4% in the minimized instances, which could affect directly in the searching step.

Another effect of minimization techniques, like *Strengthening* [1] [3], is the generation of clauses longer than the longest clause in the raw instance (i.e.: the raw MS4 instance has clauses of length 7, while the minimized instance has clauses of length 8 and 9).

Distribution of variables pre/post Minimization In Section 3, we explained that SAT variables obtained after encoding a CSP instance can be linked to three sources: to CSP decision variables, to CSP auxiliary variables and, as result of the encoding of constraints. For the Magic Square instances, the origins of variables are:

- Decision variables, from the CSP decision variables (CSP_{DV}).
- Auxiliary variables, from the CSP auxiliary variables (CSP_{AUX}).
- Auxiliary variables, from the encoding of the `alldiff()` constraints (*DIFF*).

[1] If there exist two clauses $C_1 = \{l, D\}$ and $C_2 = \{\neg l, E, D\}$, then the clause C_2 can be replaced by the resolvent $C_1 \otimes C_2 = \{D, E\}$.

– Auxiliary variables, from the encoding of full adders (*ADDER*).

Table 3 reports the total number of variables and the distribution (in percentage) of the variables by their origins. The results obtained show that preprocessing does not affect the distribution of variables but, as long as the instances grow, the percentage of auxiliary variables coming from constraint encodings increases.

Table 3. Distribution of variables pre/post minimization.

	Instance	vars.	Decision(%)	Auxiliary(%)		
			CSP_{DV}	CSP_{AUX}	DIFF	ADDER
Raw CNF	MS4	1773	12.9	0.8	45.7	40.6
	MS5	3674	11.2	0.6	39.2	49.0
	MS6	7484	9.4	0.5	31.2	59.0
	MS7	12728	8.1	0.4	26.9	64.7
Minimized CNF	MS4	675	11.1	0.7	45.7	40.6
	MS5	1372	9.7	0.3	37.1	53.0
	MS6	3092	8.2	0.4	31.2	60.1
	MS7	5381	7.1	0.4	25.7	67.6

5.3 Analysis of the Branching Behavior of a SAT Solver

This analysis focuses in the decisions made by the SAT solver, based on the origin of variables previously stated. Out interest is to determine if a smaller set of variables used available for branching can improve the solving process in SAT.

Origin of SAT Branched Variables By combining the tracing capabilities of our graph-based tool and some modifications in the Glucose solver to visualize each new branch, we determine the origins for each variable in the MS instances, allowing us to quantify which type of variable was branched the most.

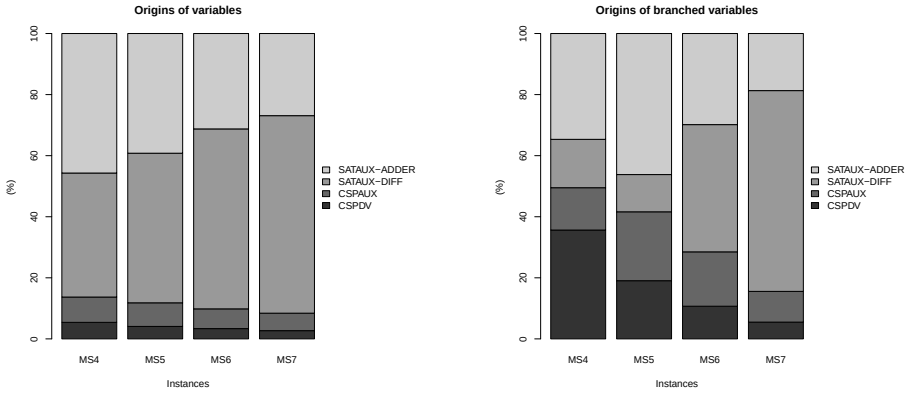
Table 4 shows the distribution by origin for all variables and for the subset of branched variables. To support the comparison, we present the same results expressed as percentages in Figures 3(a) and 3(b).

From the results obtained, we observe that even if the proportion of decision variables (CSP_{DV}) is low compared to auxiliary variables (CSP_{AUX}+ DIFF + ADDER), there is a big number of variables coming from CSP_{DV} which are branched. In average, a 82.5% of variables related to CSP_{DV} are considered during branching, making them the most branched category. For the other categories, results are 54.8% (CSP_{AUX}), 17.7% (DIFF) and 21.2% (ADDER).

Results show that variables directly linked to CSP decision variables, as they are (usually) part of multiple constraints, appear in several clauses and provide more information. In contrast, variables generated due to constraint encodings tend to appear less, their number is determined by the modeling thus, their branching utilization depends only of the CSP model provided.

Table 4. Origins of SAT variables.

	Instance	Total	Decision	Auxiliary		
			CSP_{DV}	CSP_{AUX}	DIFF	ADDER
All Variables	MS4	1773	96	147	720	810
	MS5	3674	150	284	1800	1440
	MS6	7484	252	484	4410	2338
	MS7	12728	343	729	8232	3424
Branched Vars.	MS4	202	72	28	32	70
	MS5	656	125	148	80	303
	MS6	2028	217	361	845	605
	MS7	5349	294	538	3517	1000



(a) Variables (b) Branched variables

Fig. 3. Distribution (%) of variables in CNF instances by origin.

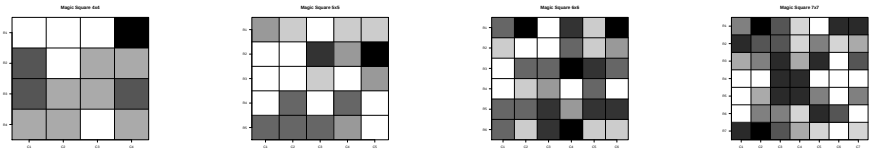
The "Solution Branch" (the set of assignments that lead to the resolution of the instances), has the same behavior. The most frequently picked variables are the ones related to CSP_{DV}, while all others are much less considered. Table 5 describes them in terms of depth (how long the branch is), and variable distribution per category. The parameter *max branches* indicates the maximum number of times that a CSP_{DV} was indirectly branched (a CSP variable is encoded in multiple SAT variables so, technically, the same CSP variable can be branched several times).

Figure 4 shows the projection of the Solution Branches in their corresponding MS grids. The colors represent how many times a CSPDV was indirectly branched; the darker the color, the higher the frequency.

These results reinforces the idea that CSP_{DV} is the most branched category so, it worths to prioritize them. Particularly for the MS, branching tends to pick variables related to the sum involving the magic number M. For example, in the grid for MS4 (Figure 4(a)), the most branched variables are linked to the linear constraint for the first row; the same for MS5 (Figure 4(b)) where the linear

Table 5. Description of Solution Branches.

Instance depth		Distribution			Max branches
	CSP_{DV}	CSP_{AUX}	DIFF	ADDER	
MS4	22	20	2	-	-	3
MS5	49	36	4	-	9	5
MS6	91	84	3	-	4	5
MS7	154	128	7	2	17	6



(a) MS4 (b) MS5 (c) MS6 (d) MS7

Fig. 4. Projection of branched variables.

constraint for the second row is the most branched. For MS6 and MS7 (Figures 4(c) and 4(d)), the most branched variables are linked to the last additions of some rows and columns. This is an interesting fact, as M is a constant and, consequently, the only which makes possible the Unit Propagation.

Sorted versus Shuffled Instances Our graph encodes to SAT following the order initially given to CSP constraints, making that CSP_{DV} variables will appear first in the CNF file. We analyze this, because studies like [2] suggests that SAT solvers are sensitive to the CNF file organization.

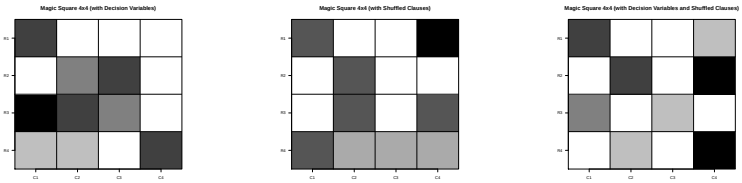
We analyze 4 versions derived from the MS4 instance: the classical one (MS4), with clauses shuffled (MS4s), branching only on CSP_{DV} variables (MS4dv) and, with clauses shuffled and branching only on CSP_{DV} variables (MS4dvs). Table 6 and Figure 5 show the results for the MS4 Solution Branches.

Table 6. Solution Branches for MS4 versions.

Instance	depth	Distribution			
		CSP_{DV}	CSP_{AUX}	DIFF	ADDER
MS4	22	20	2	-	-
MS4dv	22	22	-	-	-
MS4s	20	16	2	-	2
MS4dvs	19	19	-	-	-

The results show again the predominance of SAT_{DV} variables in the branch. The size of branches remains almost the same in the four cases, so nothing can be concluded from that.

But, observing the cases limited to branch over decision variables (MS4dv and MS4dvs), the grids projected looks more sparse than in the cases where all variables are candidate for branching. The cause for this could be the presence of the *CDCL learning processes*. The work by adding new clauses derived from the knowledge obtained after branch and propagation. It may seems, that by reducing the variables set, we are also reducing the learning speed of the solver.



(a) MS4dv (b) MS4s (c) MS4dvs

Fig. 5. Projection of branched variables for MS4.

5.4 SAT Solving for Different Branching Sets

Finally, we present the SAT solving results for different branching sets. We added two new instances (MS8 and MS9) to observe better the changes produced by the different sets.

The values reported are the runtime (in seconds), the number of propagations and the number of decision points; all of them are provided directly by the solver. The runtime has an upper limit of 2 hours (if any instance goes beyond that time, we will report ∞ values). In Figure 6, runtimes are log-scaled in order to handle the huge differences between instances.

Table 7 shows that number of decision is considerably less for the first three instances (MS 4,5,6) when branching uses all variables coming from CSP (Branching on SAT_{CSP}) and, it is the only group where MS8 and MS9 instances were solved in the time frame given. The same is observed in Figure 6.

Regarding the second group of values (Branching on SAT_{DV}), we observe a fast increment in the number of Decisions. This seems to agree with the discussion about branching and learning presented in Subsection 5.3. Moreover, if we consider that CSP_{AUX} variables belong to the magic number, then, we conclude that avoid SAT variables linked to CSP auxiliary variables reduce the chances for the Unit Propagation.

A comparison between Tables 1 and 7 show that SAT solver outperforms the resolution of the decomposed CSP model from MS7 instance onwards. In the "Standard" cases, both solvers perform similar on the small instances (MS4, MS5 and MS6), but for MS7, the CSP outperforms the SAT solver.

Table 7. SAT Results for the Magic Square.

	Instance	Runtime	Propagations	Decisions
Standard model	MS4	0.03	92944	1927
	MS5	0.11	384221	8272
	MS6	0.54	2034502	37941
	MS7	78.90	155917224	2551515
	MS8	156.83	537569994	7602086
	MS9	∞	∞	∞
Branching on SAT$_{DV}$	MS4	0.08	142316	2394
	MS5	0.12	277330	5110
	MS6	1.27	7745848	124110
	MS7	76.14	245845662	3904006
	MS8	∞	∞	∞
	MS9	∞	∞	∞
Branching on SAT$_{CSP}$	MS4	0.04	29814	536
	MS5	0.12	218608	3638
	MS6	1.22	6489299	111836
	MS7	74.99	224437398	3653155
	MS8	466.68	1176044383	16278155
	MS9	4828.31	8677420297	125249474

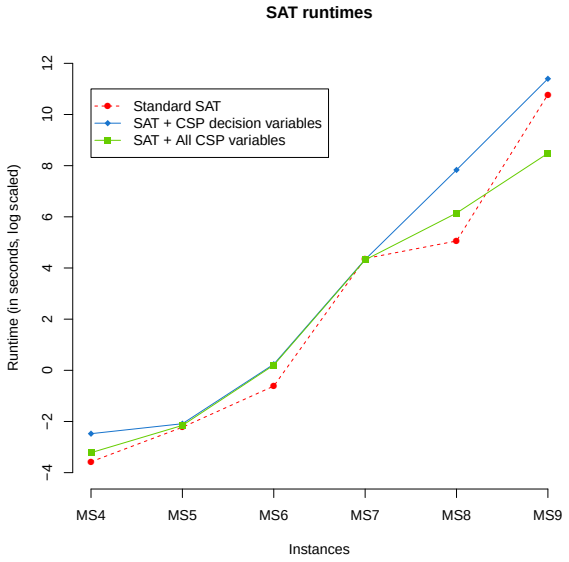


Fig. 6. SAT runtime (log scaled).

6 Conclusions and Future Work

This article analyzes the behavior of the branching heuristics of a SAT CDCL solver when CSP semantic information about their variables is added to SAT models. From the results we conclude that SAT solver intuitively branches on

variables directly linked to CSP model, despite the fact that these variables are a minority in the universe of SAT variables for a single instance. To achieve this analysis, we also propose a graph-based tool able to keep tracking of the transformation between CSP and SAT models. With it, we traced successfully the CSP origin of each SAT variable in the problem instances, giving us a CSP point of view over the SAT resolution.

Is clear that studying the transformations between CSP and SAT models opens a lot of possibilities to understand better their interactions and to improve the current techniques involved into modeling and solving. The upcoming work goes in the line of investigate if more, and which kind, of CSP semantic information could be added to a SAT model and, if it is possible to replicate a CSP-like propagation in SAT. We expect that our findings can contribute to expand the knowledge in this area.

References

1. Audemard, G., Simon, L.: Glucose, www.labri.fr/perso/lsimon/glucose/
2. Audemard, G., Simon, L.: Experimenting with small changes in conflict-driven clause learning algorithms. In: Principles and Practice of Constraint Programming. pp. 630–634 (2008)
3. Balint, A., Manthey, N.: Boosting the performance of SLS and CDCL solvers by preprocessor tuning. In: POS@ SAT. pp. 1–14 (2013)
4. Derksen, H., Eggermont, C., Van Den Essen, A.: Multimagic squares. American Mathematical Monthly 114(8), 703–713 (2007)
5. Eén, N., Sorensson, N.: Translating pseudo-boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2, 1–26 (2006)
6. Schulte, C., Lagerkvist, M., Tack, G.: Gecode, www.gecode.org
7. Walsh, T.: SAT vs CSP. In: Principles and Practice of Constraint Programming – CP 2000: 6th International Conference. pp. 441–456 (2000)