Dependent and Independent Variables in Propositional Satisfiability

Enrico Giunchiglia, Marco Maratea, and Armando Tacchella

DIST, Università di Genova
Viale Causa 13 – 16145 Genova, Italy
{enrico,marco,tac}@mrg.dist.unige.it

Abstract. Propositional reasoning (SAT) is central in many applications of Computer Science. Several decision procedures for SAT have been proposed, along with optimizations and heuristics to speed them up. Currently, the most effective implementations are based on the Davis, Logemann, Loveland method. In this method, the input formula is represented as a set of clauses, and the space of truth assignments is searched by iteratively assigning a literal until all the clauses are satisfied, or a clause is violated and backtracking occurs. Once a new literal is assigned, pruning techniques (e.g., unit propagation) are used to cut the search space by inferring truth values for other variables.
In this paper, we investigate the "independent variable selection (IVS) heuristic", i.e., given a formula on the set of variables N, the selection is restricted to a – possibly small – subset S which is sufficient to determine a truth value for all the variables in N. During the search phase, scoring and selection of the literal to assign next are restricted to S, and the truth values for the remaining variables are determined by the pruning techniques of the solver. We discuss the possible advantages and disadvantages of the IVS heuristic. Our experimental analysis shows that obtaining either positive or negative results strictly depends on the type of problems considered, on the underlying scoring and selection technique, and also on the backtracking scheme.

1 INTRODUCTION AND MOTIVATIONS

Propositional reasoning (SAT) is central in many applications of Computer Science. Several decision procedures for SAT have been proposed, along with optimizations and heuristics to speed them up. Currently, the most effective implementations are based on the Davis, Logemann, Loveland method (DLL) [Davis *et al.*, 1962]. In this method, the input formula is represented as a set of clauses, and the space of truth assignments is searched by iteratively assigning a literal until all the clauses are satisfied, or a clause is violated and backtracking occurs. Once a new literal is assigned, pruning techniques (e.g., unit propagation) are used to cut the search space by inferring truth values for other variables. Needless to say, a crucial component in every SAT solver is the heuristic used to score the literals and select the next one to assign. Every year, new heuristics are proposed and evaluated, each one tuned for a particular set of problems (see, e.g., [Li and Anbulagan, 1997, Copty *et al.*, 2001, Dubois and Dequen, 2001]).

In this paper, we investigate the "independent variable selection (IVS) heuristic", i.e., given an input formula on the set of variables N, the selection is restricted to a – possibly small – subset S which is sufficient to determine a truth value for all the variables in N. During the search phase, scoring and selection of the literal to assign next are restricted to S, and the truth values for the remaining variables are determined by the pruning techniques of the solver. The completeness of the solver is maintained because all the variables are assigned once those in S are. Moreover, the worst-case size of the search space goes down from $2^{|N|}$ to $2^{|S|}$. IVS has been proposed and used several times in combination with different scoring and selection techniques. The results have been mixed, but most of the times big benefits have been reported. In particular, in [Crawford and Baker, 1994] the IVS heuristic has been first proposed and applied to scheduling problems: no benefits are reported on these instances; in [Giunchiglia *et al.*, 1998] IVS has been applied to planning problems generated with MEDIC [Ernst *et al.*, 1997]: improvements reach 4 orders of magnitude; in [Shtrichman, 2000] and [Copty *et al.*, 2001] IVS is experimented in the context of formal verification: the authors report significant improvements on most instances. On the implementation side, the SAT solvers SIM [Giunchiglia *et al.*, 2001] and SIMO [Copty *et al.*, 2001] both incorporate the IVS heuristic along with a variety of underlying scoring and selection techniques.

We discuss the possible advantages and disadvantages of using the IVS heuristic. Our conclusion is that it is difficult to predict whether IVS will improve performances or not. Our experimental analysis shows that obtaining either positive or negative results strictly depends on the type of problems considered, on the underlying scoring and selection techniques, and also on the backtracking scheme.

The paper is structured as follows. In Section 2 we review the DLL method (Section 2.1), the concept of variable dependency (Section 2.2), and we discuss the impact that one may expect from the IVS heuristic (Section 2.3). In Section 3 we briefly describe the instances (Section 3.1) as well as the scoring and selection techniques that we experimented with (Section 3.2), and we give a detailed presentation as well as an overall summary of the results obtained (Section 3.3). We end the paper in Section 4 with some final remarks.

2 SAT, DLL AND VARIABLE DEPENDENCY

Propositional satisfiability is the task of deciding whether a given propositional formula is satisfiable or not. Most procedures do not deal with arbitrary formulas, but only with formulas in conjunctive normal form (CNF). A CNF formula is represented as a set of set of literals. A set of literals $\{l_1, \ldots, l_n\}$ $(n \geq 0)$ stands for the clause $(l_1 \vee \ldots \vee l_n)$, and a set of clauses $\{c_1, \ldots, c_m\}$ $(m \geq 0)$ stands for the CNF formula $(c_1 \wedge \ldots \wedge c_m)$. Different methods can be used to determine a satisfying assignment, i.e., a consistent set of literals entailing the set of clauses.

```
LOOK-AHEAD(Γ, U)
1 while a unit clause {l} is in Γ do
2   U ← U ∪ {l}
3   for each clause c ∈ Γ do
4     if l ∈ c then
5       Γ ← Γ \ {c}
6     else if l̄ ∈ c then
7       Γ ← (Γ \ {c}) ∪ {c \ l̄}
```

```
DLL-SOLVE(Γ, U)
1 if f = ∅ then return T
2 if ∅ ∈ f then return F
3 LOOK-AHEAD(Γ, U)
4 l ← CHOOSE-LITERAL(Γ)
5 return DLL-SOLVE(Γ ∪ {l}, U) or
            DLL-SOLVE(Γ ∪ {l̄}, U)
```

Fig. 1. The DLL method.

Our work focuses on DLL, one of the most popular solving methods. In the following, we use $l, l_1, \ldots$ to denote literals; $c, c_1, \ldots$ to denote clauses; $|l|$ to denote the variable in l; $\neg l$ to denote $\neg|l|$ if $l = |l|$, and $|l|$ otherwise.

2.1 DLL

The function DLL-SOLVE in Figure 1 is the pseudo-code of the DLL method. The parameters of DLL-SOLVE are a set of clauses Γ, and an assignment U. Initially, Γ is the set of clauses corresponding to the input formula, and U is the empty set. DLL-SOLVE returns "T" exactly when Γ is satisfiable, and "F" otherwise.

Most implementations of the DLL method differ in the scoring and selection techniques used by CHOOSE-LITERAL, and in the kind of simplifications performed by LOOK-AHEAD. In particular, the LOOK-AHEAD procedure in Figure 1, simplifies the set of clauses by detecting and propagating unit clauses only. This technique is also known as *Boolean constraint propagation* (BCP). Another popular simplification technique is *monotone literal fixing* (MLF): For each literal l, if $\neg l$ does not belong to any clause in Γ, then we can add $\{l\}$ to the set of clauses. The addition of MLF preserves the correctness and completeness of the method.

Another difference among DLL implementations is the backtracking scheme. The function DLL-SOLVE in Figure 1 implements *chronological backtracking*. According to this scheme, once an empty clause is found, the search resumes from the latest literal selected by CHOOSE-LITERAL. In DLL-SOLVE this is accomplished by a disjunction of two recursive calls (line 5). Most modern implementations of DLL (see, e.g., [Giunchiglia *et al.*, 2001]) feature much more advanced backtracking schemes, the most popular being backjumping and learning. *Backjumping* enables DLL to backtrack and skip over literals that are not responsible for the generation of an empty clause; *Learning* augments the input formula with clauses inferred during the backtrack phase. For a detailed description of these schemes see, e.g., [Giunchiglia *et al.*, 2001]. Here it is enough to say that backjumping and learning have had a substantial impact in practice and are implemented in most state-of-the-art DLL solvers.

2.2 Variable Dependency

The idea behind variable dependency in DLL is simple. A variable x is *dependent* on a set of independent variables $x_1, \ldots, x_n$ if for any assignment to $x_1, \ldots, x_n$ the value of x becomes determined by LOOK-AHEAD. Thus, the notion of variable dependency is defined with respect to a given set of variables and a specific LOOK-AHEAD procedure. For example, in the formula:

$$\{\{\neg l \vee l_1 \vee \ldots l_n\}, \{\neg l_1 \vee l\}, \ldots, \{\neg l_n \vee l\}\}$$

where $l, l_1, \ldots, l_n$ are distinct literals, we can say that $|l|$ depends on $|l_1|, \ldots, |l_n|$ if LOOK-AHEAD enforces BCP, and each $|l_i|$ depends on $|l|$ if LOOK-AHEAD enforces MLF. Our working hypotheses for the remainder of the paper are that LOOK-AHEAD is limited to BCP as in Figure 1, and that the set of independent variables is known a priori in all the problems that we consider.

2.3 DLL and Variable Dependency

If S is a set of independent variables, it is rather easy to modify CHOOSE-LITERAL to enforce IVS, i.e., to restrict scoring and selection to S. Now the question is: what can we expect by using IVS? For one thing, the unrestricted CHOOSE-LITERAL can simulate the restricted one, but the converse is not true. If the input formula φ is satisfiable, this is not necessarily a problem: they are both allowed to pick a set of literals satisfying φ. On the other hand, if the input formula is unsatisfiable, we may witness a substantial degradation of DLL performances when using the IVS heuristic. For example, let φ be an unsatisfiable formula on the set of variables S. Assume that refuting φ with DLL requires a search tree of exponential size (the existence of such formulas is discussed, e.g., in [Urquhart, 1995]). Now let ψ be the following formula on the set of variables $N = S \cup \{a, b, c\}$:

$$(\varphi \leftrightarrow a) \wedge (a \leftrightarrow b) \wedge (a \leftrightarrow c) \wedge (b \leftrightarrow \neg c).$$

Clearly, ψ itself is unsatisfiable and the variables a, b, c are dependent on the set of variables S. If we try to have DLL refute ψ by enabling the IVS heuristic, then we are bound to explore a search tree of exponential size. At the end of each branch we determine a truth value for a, and then backtrack occurs because of an inconsistency found by running LOOK-AHEAD. On the other hand, it is sufficient for an unrestricted CHOOSE-LITERAL to pick any of the variables in the set $\{a, b, c\}$ in order to quickly refute ψ.

Notwithstanding the above considerations that may suggest to abandon the idea of restricting CHOOSE-LITERAL, in practice it is easy to see how exponential improvements can be obtained with the IVS heuristic. For example, given a formula φ on a set S of variables, we can construct a superformula ψ of φ that introduces denumerately many new variables, all depending on S. If we introduce exponentially many new such variables and try to solve ψ, the first call to an unrestricted CHOOSE-LITERAL is enough to cause a substantial worsening with respect to a restricted one.

Statistic	DES	Parity	BMC	Pretolani	Planning
Min	0.01	0.01	0.05	0.35	0.67
$\mathbf{Q_1}$	0.01	0.02	0.10	0.34	0.68
$\mathbf{Q_2}$	0.03	0.03	0.18	0.37	0.70
$\mathbf{Q_3}$	0.06	0.05	0.24	0.37	0.71
Max	0.18	0.13	0.40	0.46	0.74

Table 1. Statistics on the ratio $|S|/|N|$.

3 EXPERIMENTAL RESULTS

3.1 Test set

The test set used to evaluate the IVS heuristic consists of 157 CNF formulas, 102 satisfiable and 55 unsatisfiable, yielding an overall 64% chance of a satisfiable instance. Some of these instances are also known to be challenging for currently available SAT solvers, but selecting hard instances was not our focus. In assembling the test set, we privileged problems that have already been used to compare SAT solvers; we tried to maximize the number of different problem classes, and the number of problems in each class. Our choice was constrained by the availability of instances for which we are able to determine a set of independent variables. In particular, we choosed:

- 32 Data Encryption Standard (DES) problems, see [Massacci and Marraro, 2000, Li, 2000];
- the "famous" 30 parity problems, see, e.g., [Selman *et al.*, 1997];
- 34 instances of bounded model checking (BMC), see [Biere *et al.*, 1999, Li, 2000];
- 24 Pretolani problems, see, e.g., [Li, 2000].
- 37 planning problems generated by BlackBox [Kautz and Selman, 1998];

For the sake of our analysis, the most representative parameter is the $|S|/|N|$ ratio, i.e., the fraction of the set of total variables N which is also in the set S of independent variables.

Table 1 gives an idea of how the ratio $|S|/|N|$ is distributed among the instances in each of the above problem classes. We remind that the p%-percentile is the value x such that p% of the observed data is smaller than x. In Table 1: Q_1 denotes the 25%-percentile, Q_2 the 50%-percentile, and Q_3 the 75%-percentile of the observed ratios; "Min" and "Max" denote, respectively, the minimum and the maximum of the observed ratios. This set of statistics is known as the "five-number summary" and is most useful for comparing distributions [Moore and McCabe, 1993]. From Table 1 we can see that the parameter $|S|/|N|$ is distributed quite differently across the problem classes considered. For instance, while $|S|/|N|$ is almost normally distributed around 0.18 from a minimum of 0.05 to a maximum of 0.40 in BMC problems, in DES and Parity instances the distribution is biased towards small values of $|S|/|N|$ since the 75% of the data is smaller than 0.06 for DES and 0.05 for Parity problems, but the maximum ratio is 0.18 for DES and 0.13 for Parity problems.

3.2 DLL implementation

As we mentioned in Section 1, the SAT solver SIM supports the IVS heuristic. Moreover, SIM comes with different implementations of CHOOSE-LITERAL that can be selected from the command line. Using compile-time options, it is also possible to augment SIM with backjumping and learning. Therefore, SIM provided the ideal test bench for our purposes. In our experiments, we run SIM with 5 different implementations of CHOOSE-LITERAL, with and without backjumping and learning. For each version of CHOOSE-LITERAL we run the default implementation as well as the one with the IVS heuristic, for a total of 20 different configurations tested. Unless explicitly mentioned, the version of SIM used to produce the data is the one with chronological backtracking. The implementations of CHOOSE-LITERAL that we evaluated are:

- MOMS (M), introduced in [Pretolani, 1993], which prefers variables that occur frequently in the shortest clauses;
- Jeroslow-Wang (JW), see [Jeroslow and Wang, 1990], where the occurrences of variables in short clauses are exponentially better than those in long clauses;
- Böhm (B), discussed in [Buro and Buning, 1992], which considers occurrences in clauses of any length and, in case of ties, prefers variables occurring frequently in short clauses;
- SATZ (S), as explained in [Li and Anbulagan, 1997], which features a complex scoring mechanism based on BCP and a modified version of JW;
- Unitie0 (U0), introduced by [Copty *et al.*, 2001] under the name "Unit", which prefers variables producing the highest simplification with BCP.

In all the cases above, using the IVS heuristic amounts to restricting the scoring and selection process to the variables in S. To distinguish between the unrestricted CHOOSE-LITERAL and the one with the IVS heuristic, we use "*" as a suffix for the latter. For instance, "M*" means MOMS scoring with IVS enabled.

3.3 Effects of the IVS Heuristic

Tables 2 to 6 report the results of our experimental analysis using the instances and the configurations of SIM described in the previous subsections. The ordering of the tables is in accordance with the value of Q_2 for each problem class, from the lowest (DES instances) to the highest (Planning instances). Indeed, considering the discussion in Section 2.3, we expected that using the IVS heuristic would produce greater benefits for problems with small values of $|S|/|N|$.

Each line in the tables reports the following data:

- the configuration of SIM, e.g., "M" and "M*" for MOMS;

– the cumulative sum $C(t)$ of the problems solved by the configuration within time t, for $t = \{1, 2, 4, \ldots, T_f\}$; T_f is 1200 seconds;
– the root mean square (RMS) of $C(t)$ calculated as:

$$\mathrm{RMS}[C(t)] = \sqrt{\frac{\sum_{s=1}^{T_f/T} C(sT)^2}{T_f/T}}$$

where T is a sampling constant that divides the range $[0; T_f]$ into intervals of equal length; in our calculations $T = 12$.

The value RMS$[C(t)]$ summarizes the performances of a given configuration, and enables the comparison among different configurations. Intuitively, it represents the weighted distance of $C(t)$ from the "worst possible configuration" $C_0(t) = 0$, i.e., the configuration which cannot solve any problem within T_f. The use of RMS$[C(t)]$ privileges the configurations that converge quickly and solve many problems. RMS$[C(t)]$ is going to be small for configurations that either converge slowly or saturate at a small number of problems. This captures precisely our intuition of a "bad" behavior of a configuration on a problem class. [1]

Heur	**RMS**	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	$\mathbf{T_f}$
M	15.81	3	3	5	6	10	13	14	16	16	16	16
M*	15.90	3	4	5	9	12	14	15	16	16	16	16
B	15.83	3	4	5	6	9	13	15	16	16	16	16
B*	15.91	4	5	5	8	12	15	16	16	16	16	16
JW	15.66	2	2	3	4	4	8	13	15	16	16	16
JW*	15.83	2	2	4	4	7	13	14	16	16	16	16
S	16.23	16	16	16	16	16	16	16	16	16	16	17
S*	16.22	16	16	16	16	16	16	16	16	16	16	17
U0	23.66	16	18	21	22	22	22	22	22	23	24	24
U0*	16.00	16	16	16	16	16	16	16	16	16	16	16

Table 2. Data Encryption Standard, 32 problems, all satisfiable.

In Table 2 the data about DES problems is reported. Notice that none of the configurations is able to solve all the problems in the class. Looking at the statistics for $|S|/|N|$ in Table 1, we see small ratios for these instances, always less than 0.18. Nevertheless, introducing the IVS heuristic does not help in speeding up the convergence of any configuration, modulo some minor improvements on MOMS, Böhm and Jeroslow-Wang. The null impact on SATZ, is contrasted by a sharp worsening of the Unitie0 heuristic: "U0*" solves 1/3 less problems than the unrestricted "U0" configuration.

[1] All the experiments run on a network of identical workstations equipped with PentiumIII 600Mhz, 128MB of RAM, and `Linux Suse ver. 6.2`. SIM is compiled with `gcc 2.95.2`; time is measured in CPU seconds.

Heur	RMS	1	2	4	8	16	32	64	128	256	512	T_f
M	19.99	10	10	12	17	19	20	20	20	20	20	20
M*	20.00	12	13	15	19	20	20	20	20	20	20	20
B	19.99	11	13	13	16	20	20	20	20	20	20	20
B*	20.00	12	15	20	20	20	20	20	20	20	20	20
JW	19.99	13	16	19	20	20	20	20	20	20	20	20
JW*	20.00	10	13	15	17	18	20	20	20	20	20	20
S	19.99	12	15	16	18	20	20	20	20	20	20	20
S*	19.99	12	15	16	18	20	20	20	20	20	20	20
U0	19.63	11	11	12	12	14	16	18	19	19	20	20
U0*	20.00	14	17	19	20	20	20	20	20	20	20	20

Table 3. Parity, 30 problems, all satisfiable.

In Table 3 we report the data about Parity problems. As for the DES class, none of the configurations is able to solve all the problems in the class. This is to be expected, since these problems have been shown to be very challenging for the current state-of-the-art SAT solver. On the other hand, the statistics for the $|S|/|N|$ ratio in Table 1 tell us that the percentage of independent variables is quite small (13% at most). Indeed, introducing the IVS heuristic helps a little in speeding up the convergence of some configurations, namely MOMS, Böhm, Unitie0, but it has a negative impact on Jeroslow-Wang and a null impact on SATZ configurations. Moreover, the IVS heuristic does not help in solving more problems (see the T_f column). Parity problems are thus a clear example of how the IVS heuristic can be successful for some configurations and unsuccessful for others.

Heur	RMS	1	2	4	8	16	32	64	128	256	512	T_f
M	26.11	9	11	12	13	16	16	18	19	21	26	30
M*	29.71	11	12	15	16	18	19	20	24	26	32	32
B	25.70	8	10	12	13	16	16	18	19	21	26	30
B*	29.67	11	12	14	16	18	19	20	24	26	32	32
JW	17.10	9	9	10	11	13	13	14	15	17	17	18
JW*	28.58	11	14	16	17	18	20	21	23	25	28	32
S	21.22	10	12	12	14	14	16	17	17	20	21	24
S*	21.20	10	12	12	14	14	16	17	17	20	21	24
U0	17.95	9	9	10	12	12	13	14	15	17	18	20
U0*	21.96	11	12	13	13	14	16	18	18	20	22	25

Table 4. Bounded Model Checking, 34 problems, all unsatisfiable.

In Table 4 the data about BMC problems is reported. In this problem class (see Table 1) the median $|S|/|N|$ ratio is 0.18, and it is higher than the maximum ratio observed on DES and Parity problems. Nevertheless, with the only exception of SATZ scoring, introducing the IVS heuristic helps in solving more problems (see the T_f column), and also in speeding up convergence (see the RMS column). The data is in accordance with the good results reported on similar problems in the literature (see, e.g., [Shtrichman, 2000] and [Copty *et al.*, 2001]).

Heur	**RMS**	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	$\mathbf{T_f}$
M	8.68	4	5	5	5	7	7	8	8	8	9	9
M*	8.76	6	8	8	8	8	8	8	8	8	9	9
B	8.65	3	3	3	3	4	5	6	6	6	6	6
B*	14.75	11	12	12	12	12	12	12	12	12	12	12
JW	14.45	11	13	14	14	14	14	14	14	14	14	15
JW*	14.55	13	14	14	14	14	14	14	14	14	14	15
S	12.55	8	10	10	11	11	11	11	11	12	13	13
S*	9.15	7	7	7	8	8	8	8	8	9	9	10
U0	14.22	10	11	13	14	14	14	14	14	14	14	15
U0*	14.66	12	13	14	14	14	14	14	14	14	15	15

Table 5. Pretolani, 24 problems, 12 satisfiable, 12 unsatisfiable.

Table 5 reports about Pretolani instances. On this class we see again some improvement brought by the IVS heuristic. The distribution of $|S|/|N|$ is concentrated in the interval $[0.35; 0.46]$ with a median of 0.37. Notwithstanding the fairly big number of variables left to examine, in at least one configuration, namely Böhm, introducing the IVS heuristic enables to solve more problems within T_f. In some configurations, namely MOMS, Jeroslow-Wang and Unitie0, the convergence of the configurations featuring IVS is quicker than their unrestricted counterparts. On the other hand, SATZ performances worsen with the introduction of IVS. Indeed, SATZ with IVS does not examine all the variables and this amounts to a loss of accuracy in the selection of the next variable. Such loss is not compensated by the speed gain obtained by examining an approximate 40% of the variables only.

We conclude our report with the data about Planning problems in Table 6. From the table we can immediately grasp the inadequacy of non-BCP based heuristics on this problem class. MOMS, Böhm and Jeroslow-Wang fail to be really competitive, and the improvements introduced by the IVS heuristic are quite marginal, with the only exception of MOMS. SATZ and Unitie0 configurations are much more performant, but again the IVS heuristic is not able to really improve them in any sense.

In Table 7 we summarize the results of our experiments in a compact way, with the following notation. Each line corresponds to a configuration, and the comparison is between the unrestricted version and the one featuring the IVS heuristic. There are two blocks of results in the table. The upper block summarizes the results obtained using SIM with chronological backtracking, while

Heur	RMS	1	2	4	8	16	32	64	128	256	512	T_f
M	5.63	0	2	3	4	4	5	5	5	5	5	7
M*	7.80	0	2	5	5	5	6	6	6	6	7	9
B	2.04	0	0	0	0	0	0	2	2	2	2	3
B*	2.33	0	0	0	0	0	0	2	2	2	2	3
JW	0.97	0	0	0	0	0	0	0	1	1	1	1
JW*	0.97	0	0	0	0	0	0	0	1	1	1	1
S	27.99	10	11	13	17	19	20	21	24	27	28	31
S*	27.99	10	11	13	17	19	20	21	24	27	28	31
U0	27.79	7	10	10	19	22	23	24	25	26	28	30
U0*	26.92	5	11	16	20	21	21	22	22	24	27	30

Table 6. Planning (BlackBox), 37 problems, 28 satisfiable, 9 unsatisfiable.

Heur	DES	Parity	BMC	Pretolani	Planning
M	$+$	$+$	$++$	$+$	$+$
B	$+$	$+$	$++$	$+++$	$+$
JW	$+$	$+$	$+++$	$+$	$=$
S	$-$	$=$	$-$	$--$	$=$
U0	$---$	$+$	$++$	$+$	$-$
M	$+$	$+$	$+$	$--$	$+$
B	$+$	$+$	$+$	$-$	$+$
JW	$+$	$-$	$++$	$--$	$+$
S	$-$	$=$	$-$	$-$	$=$
U0	$-$	$+$	$++$	$-$	$+$

Table 7. Evaluation of the IVS heuristic.

the data in the lower block is obtained using SIM augmented with backjumping and learning. Each column corresponds to a problem class. To compute the data in the Table, we use the RMS values for each configuration. Given $C(t)$ and $R = \mathrm{RMS}[C(t)]$ for an unrestricted configuration, we denote with $C^*(t)$ and $R^* = \mathrm{RMS}[C^*(t)]$ the corresponding parameters for the same configuration with the IVS heuristic. With these definitions in mind, a "+"(resp., "−") in Table 7 means that $R^* > R$ (resp. $R^* < R$), and a "=" means that $R^* = R$. To quantify how much the RMS changes going from $C(t)$ to $C^*(t)$, let P be the total number of problems in the class and r be the ratio

$$r = \frac{|R^* - R|}{P}$$

In Table 7, given $\odot \in \{+, -\}$, a single "$\odot$" means that $r \leq 0.1$, a "$\odot\odot$" means that $0.1 < r \leq 0.2$, and a "$\odot \odot \odot$" means that $r > 0.2$. For instance, the "+ +" corresponding to BMC and U0 in Table 7 means that U0* is 10% to 20% better than U0. The upper block of Table 7 evidentiates that both the problem class and the underlying scoring technique determine the behavior of IVS heuristic, and the lower block of Table 7 evidentiates that also the backtracking scheme should be included among the factors that make up for IVS effectiveness.

Overall, on the basis of the results in Table 7, we can conclude that whether IVS improves performances or not strictly depends on the type of problems considered, on the underlying scoring and selection technique, and also on the backtracking scheme. Still, according to the data in Table 7, IVS introduces some benefits on average: in the Table there are 40 "+", and 19 "−".

4 CONCLUSIONS

In this paper, we have investigated the "independent variable selection (IVS) heuristic". We have discussed the possible advantages and disadvantages of using IVS that one may expect a priori, and we concluded that determining whether the heuristic will improve the performances or not is far from being obvious. The experimental analysis that we presented confirms this intuition. We showed that obtaining either positive or negative results strictly depends on the type of problems considered, on the underlying scoring and selection techniques, and on the backtracking scheme. Nevertheless, the potential of the heuristic weighted against its simplicity is fairly big, and this suggests that, whenever possible, IVS should be included in the experimental assessment of SAT instances.

References

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, 1999.

[Buro and Buning, 1992] M. Buro and H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.

[Cimatti *et al.*, 2002a] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. CAV*, 2002. To appear.

[Cimatti *et al.*, 2002b] A. Cimatti, E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Integrating BDD-based and SAT-based symbolic model checking. In A. Armando, editor, *Proceedings of the 4rd International Workshop on Frontiers of Combining Systems (FroCoS 2002)*, volume 2309 of *Lecture Notes in Computer Science*, pages 49–56. Springer-Verlag, 2002.

[Copty *et al.*, 2001] Fady Copty, Limor Fix, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Vardi. Benefits of bounded model checking at an industrial setting. In *Proc. 13th International Computer Aided Verification Conference (CAV)*, 2001.

[Crawford and Baker, 1994] James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1092–1097, Seattle, Washington, USA, August 1994. AAAI Press/MIT Press.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[Dubois and Dequen, 2001] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In Bernhard Nebel, editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 248–253, San Francisco, CA, August 4–10 2001. Morgan Kaufmann Publishers, Inc.

[Ernst *et al.*, 1997] Michael Ernst, Todd Millstein, and Daniel Weld. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*, 1997.

[Giunchiglia and Sebastiani, 1999] E. Giunchiglia and R. Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In Evelina Lamma and Paola Mello, editors, *Proceedings of AI*IA'99: Advances in Artificial Intelligence, LNAI 2175*, pages 84–94. Springer Verlag, 1999.

[Giunchiglia *et al.*, 1998] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proc. AAAI*, 1998.

[Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and Davide Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001), LNAI 2083*, 2001.

[Jeroslow and Wang, 1990] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[Kautz and Selman, 1998] Henry Kautz and Bart Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98*, 1998.

[Li and Anbulagan, 1997] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[Li, 2000] Chu Min Li. Integrating equivalence reasoning into Davis-Putnam procedure. In *Proc. AAAI*, 2000.

[Massacci and Marraro, 2000] Massacci and Marraro. Logical cryptanalysis as a SAT problem. *JAR: Journal of Automated Reasoning*, 24, 2000.

[Moore and McCabe, 1993] D. S. Moore and G. P. McCabe. *Introduction to the Practice of Statistics*. W. H. Freeman and Co., 1993.

[Pretolani, 1993] Daniele Pretolani. *Satisfiability and Hypergraphs*. PhD thesis, Università di Pisa, 1993.

[Selman *et al.*, 1997] Bart Selman, Henry Kautz, and David McAllester. Ten challenges in propositional reasoning and search. In *Proc. IJCAI-97*, pages 50–54, 1997.

[Shtrichman, 2000] O. Shtrichman. Tuning SAT checkers for bounded model-checking. In *Proc. 12th International Computer Aided Verification Conference (CAV)*, 2000.

[Urquhart, 1995] Alasdair Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, December 1995.