Dependent and Independent Variables in Propositional Satisfiability

Enrico Giunchiglia, Marco Maratea, Armando Tacchella

{enrico, marco, tac}@dist.unige.it

MRG-LAB DIST, Università di Genova

Problem: propositional satisfiability

A *literal* l is a proposition p or its negation $\neg p$.

Given $l_1, ..., l_k$ literals, a *clause* is $l_1 \vee ... \vee l_k$.

Given $c_1, ..., c_m$ clauses, a CNF formula is $c_1 \wedge ... \wedge c_m$.

An *assignment*, or *valuation* v, is a partial function from the propositions to $\{T, F\}$, where T and F are the truth values *true* and *false*. We can extend the definition of v in the natural way to assign truth values to literals, clauses and formulas.

Given a CNF formula φ, we define the *propositional satisfiability problem (SAT)*:

Does there exist an assignment to the propositions in φ such that φ is true?

Solving the SAT problem

Davis-Logemann-Loveland algorithm:

function DLL-Solve(Γ, U)
    if Γ = $\emptyset$ **then return** T
    if $\emptyset \in \Gamma$ **then return** F
    LookAhead(Γ, U)
    l := *ChooseLiteral*(Γ)
    return *DLL-Solve*($\Gamma \cup l$, U) **or**
            DLL-Solve($\Gamma \cup \neg l$, U)

Solving the SAT problem

Davis-Logemann-Loveland algorithm:

function DLL-Solve(Γ, U)

if $\Gamma = \emptyset$ **then return** T

if $\emptyset \in \Gamma$ **then return** F

LookAhead(Γ, U)

l := *ChooseLiteral*(Γ)

return *DLL-Solve*($\Gamma \cup l$, U) **or**

DLL-Solve($\Gamma \cup \neg l$, U)

Solving the SAT problem

Davis-Logemann-Loveland algorithm:

function DLL-Solve(Γ, U)

  if $\Gamma = \emptyset$ **then return** T

  if $\emptyset \in \Gamma$ **then return** F

  LookAhead(Γ, U)

  l := *ChooseLiteral*(Γ)

  return *DLL-Solve*($\Gamma \cup l$, U) **or**

    DLL-Solve($\Gamma \cup \neg l$, U)

Structure of the talk

- Dependent and independent variables
- Experimental analysis
- Conclusions and future work

Dependent and Independent variables(1)

Independent variables are a subset of the initial variables such that the others (called *dependent*) are determined once the values of the independent are.

Given the set of variables N and the set of independent variables S:

- the completeness is maintained because all the variables get assigned once the variables in S are
- the(worst case) size of the search space goes down from $2^{|N|}$ to $2^{|S|}$

Dependent and Independent variables(2)

The differences between dependent and independent variables arise from:

- the structure of the problems, e.g., in circuital problems it is natural to define input and state variables as independent and output variables as dependent
- the translation from non-clausal problems(natural representation) to clausal problems(adequate for SAT solvers) when variables are added to the formula

Dependent and Independent variables: implementation

The implementation in SIM consists of two parts:

- offline, with the identification of the independent variables and their explicit indication in the input instance
- online, restricting the heuristic to choose from the pool of the independent variables: independent variable selection(IVS) heuristic

Dependent and Independent variables: offline implementation

The purpose of the offline implementation is to identify the independent variables from the initial variables of the formula. There are different ways to extract independent variables:

- Bounded Model Checking: variables are divided into independent and dependent according to whether they are input, state or output variables
- Parity Learning, Dubois and Pretolani: the definitions in the formula are taken into account and used to identify independent variables
- Data Encryption Standard: the independent variables are defined statically, i.e., they are the bits of the cryptographic key
- Planning: variables are divided into independent and dependent according to whether they are action or fluent variables

Dependent and Independent variables: ratio

Statistic	BMC	Parity	Planning	DES	Pretolani
Min	0.05	0.01	0.67	0.01	0.35
Q1	0.10	0.02	0.68	0.01	0.34
Median	0.18	0.03	0.70	0.03	0.37
Q3	0.24	0.05	0.71	0.06	0.37
Max	0.40	0.13	0.74	0.18	0.46

Experimental analysis: assessment

For the experimental analysis we have used:

- 4 versions of SIM: plain and with backjumping and learning, with and without IVS heuristic
- 6 problem categories: Bounded Model Checking, Parity Learning, Dubois, Pretolani, Planning, Data Encryption Standard
- 5 heuristics: Moms(M), Boehm(B), Jeroslow-Wang(JW), Satz(S), Unitie0(U0)
- tests runned on Pentium III 600MHz, 128MB RAM

Experimental analysis: Bounded Model Checking

heuristic	0.5	1	2	4	8	16	32	64	128	256	512	1200
M	8	9	11	12	13	16	16	18	19	21	26	30
M*	8	11	12	15	16	18	19	20	24	26	32	32
B	8	8	10	12	13	16	16	18	19	21	26	30
B*	8	11	12	14	16	18	19	20	24	26	32	32
JW	9	9	9	10	11	13	13	14	15	17	17	18
JW*	10	11	14	16	17	18	20	21	23	25	28	32
S	10	10	12	12	14	14	16	17	17	20	21	24
S*	10	10	12	12	14	14	16	17	17	20	21	24
U0	7	9	9	10	12	12	13	14	15	17	18	20
U0*	9	11	12	13	13	14	16	18	18	20	22	25

Experimental analysis: Bounded Model Checking

heuristic	0.5	1	2	4	8	16	32	64	128	256	512	1200
M	8	9	11	12	13	16	16	18	19	21	26	30
M*	8	11	12	15	16	18	19	20	24	26	32	32
B	8	8	10	12	13	16	16	18	19	21	26	30
B*	8	11	12	14	16	18	19	20	24	26	32	32
JW	9	9	9	10	11	13	13	14	15	17	17	18
JW*	10	11	14	16	17	18	20	21	23	25	28	32
S	10	10	12	12	14	14	16	17	17	20	21	24
S*	10	10	12	12	14	14	16	17	17	20	21	24
U0	7	9	9	10	12	12	13	14	15	17	18	20
U0*	9	11	12	13	13	14	16	18	18	20	22	25

Experimental analysis: Data Encryption Standard

heuristic	0.5	1	2	4	8	16	32	64	128	256	512	1200
M	1	3	3	5	6	10	13	14	16	16	16	16
M*	2	3	4	5	9	12	14	15	16	16	16	16
B	2	3	4	5	6	9	13	15	16	16	16	16
B*	2	4	5	5	8	12	15	16	16	16	16	16
JW	0	2	2	3	4	4	8	13	15	16	16	16
JW*	1	2	2	4	4	7	13	14	16	16	16	16
S	16	16	16	16	16	16	16	16	16	16	16	17
S*	16	16	16	16	16	16	16	16	16	16	16	17
U0	16	16	18	21	22	22	22	22	22	23	24	24
U0*	14	16	16	16	16	16	16	16	16	16	16	16

Experimental analysis: Data Encryption Standard

heuristic	0.5	1	2	4	8	16	32	64	128	256	512	1200
M	1	3	3	5	6	10	13	14	16	16	16	16
M*	2	3	4	5	9	12	14	15	16	16	16	16
B	2	3	4	5	6	9	13	15	16	16	16	16
B*	2	4	5	5	8	12	15	16	16	16	16	16
JW	0	2	2	3	4	4	8	13	15	16	16	16
JW*	1	2	2	4	4	7	13	14	16	16	16	16
S	16	16	16	16	16	16	16	16	16	16	16	17
S*	16	16	16	16	16	16	16	16	16	16	16	17
U0	16	16	18	21	22	22	22	22	22	23	24	24
U0*	14	16	16	16	16	16	16	16	16	16	16	16

Experimental analysis: further analysis

Our results show that the IVS heuristic seems to perform well on reachability analysis problems(BMC and Planning) and Parity Learning problems, and not so well on other types of problems(Dubois, Pretolani and Data Encryption Standard).

New measures in the next slide:

- Root Min Square : $\sqrt{\frac{\sum_{s=1}^{T_f/T} X_s^2 - Y_s^2}{T_f/T}}$, with T_f = 1200 and T = 12
- version with IVS heuristic solves much problems respect to the basic version
  - "$+$" : from 0% to 10%
    "$++$" : from 10% to 20%
    "$+++$" : major than 20%
  - we can extend this meaning to "$-$"

Experimental analysis: global results for plain version

heuristic	BMC	Parity	DES	Planning	Pretolani
M	++	+	+	+	+
B	++	+	+	+	+++
JW	+++	+	+	=	+
S	—	=	—	=	——
U0	++	+	— — —	—	+

Experimental analysis: global results with bj e learning

heuristic	BMC	Parity	DES	Planning	Pretolani
M	+	+	+	+	— —
B	+	+	+	+	—
JW	++	—	+	+	— —
S	—	=	—	=	—
U0	++	+	—	+	—

Experimental analysis: summary

Our experimental analysis shows that the improvement of the performances when using IVS heuristic strictly depends on:

- the type of problem
- the splitting heuristic
- the backtracking schema

Conclusions

Using our particular strategy, i.e., splitting only on independent variables and determining the dependent with the pruning techniques of the solver, we shed new light on the claim:
“It is fairly easy to empirically demonstrate that identification of dependent variables improves the performances of systematic search”(Kautz, Selman and McAllester, “Ten Challenges in Propositional Reasoning and Search”, IJCAI, 1997).

Future work

In the future we will explore:

- Cut the numbers of independent variables using the definitions in the formulas
- Better understanding of the structure of the problems to guide the search
- Define new strategies to use the information about the variables
- Include our strategy in a state-of-the-art SAT solver

References and applications

- SIM is integrate in NuSMV, a model checker built at the University of Trento
- SIMO, a C++ version of SIM, is an ongoing project. SIMO participated to the recent SAT Competition in Cincinnati(http://www.satlive.org/SATCompetition)
- Learn more about SIM and SIMO at the STAR project homepage available from: http://www.mrg.dist.unige.it/star