Evaluating Search Strategies and Heuristics for Efficient Answer Set Programming

E. Giunchiglia and **M. Maratea**

STAR-Lab
University of Genova, Italy

Milano, AI*IA 2005

Answer Set Programming

Answer Set (stable model) Programming is a new programming paradigm proposed by Marek, Truszczynski and Niemela in 1999.

It is a form of declarative programming. It is based on logic rules and on the answer set semantic of Prolog proposed by Gelfond and Lifschitz in 1988.

In answer set programming (ASP) we obtain the answers by declaring the properties of the answers, by the mean of logic rules.

ASP: Logic rules for 3-colorability problem

```
% Simple graph containing 3 nodes and 3 edges:
% edges between nodes 1 and 2, 2 and 3, 3 and 1.
node(1).  node(2).  node(3).
edge(1,2).  edge(2,3).  edge(3,1).
% Declaration of three colors
col(red).  col(green).  col(blue).
% A node has some color
color(X,red) v color(X,green) v color(X,blue) :- node(X).
% Neighboring nodes should not have the same color
:- edge(X,Y), color(X,C), color(Y,C), col(C).
```

Logic programs

A *(logic) program* Π is a finite set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n \quad (1)$$

where P is the set of atoms in Π, $A_0 \in P \cup \{\bot\}$,
$\{A_1, \ldots, A_n\} \subseteq P$.
A_0 is the *head* of the rule. $\{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\}$ is the *body* of the rule. *not* is the "negation as failure" operator, and $\bot$ stands for *False*.
$Comp(\Pi)$ consists of formulas of the type

$$A_0 \equiv \bigvee (A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n)$$

for each symbol in $P \cup \{\bot\}$. In the equation, the disjunction extends over all rules (1) in Π with head A_0.

What is an answer set?

Consider first Π in which $m = n$, when rules does not have "*not*" operator.
Let X be a set of atoms.
We say that X is *closed* under Π if for every rule in Π, $A_0 \in X$ whenever $\{A_1, \ldots, A_m\} \subseteq X$.
We say that X is an answer set for Π if X is the smallest set closed under Π.
Consider now the general case $n > m$.
The reduct Π^X of Π related to X is the set of rules

$$A_0 \leftarrow A_1, \ldots, A_m$$

such that $X \cap \{A_{m+1}, \ldots, A_n\} = \emptyset$.
We say that X is an answer set for Π if X is an answer set for Π^X.
Given a (logic) program Π, to find if it has a solution is an NP-complete problem.

ASP: Examples

Given the atoms p, q and r

1. Be Π_1: $$\begin{aligned} p &\leftarrow \ \ not \ q \\ q &\leftarrow \ \ not \ r \end{aligned}$$
The only AS is $\{q\}$

2. Be Π_2: $$\begin{aligned} p &\leftarrow \ \ not \ q \\ q &\leftarrow \ \ not \ p \end{aligned}$$
The ASs are $\{p\}$ and $\{q\}$

3. Be Π_3: $$p \leftarrow \ \ not \ p$$
Π_3 does not have AS.

Applications of ASP

So far, answer set programming has been used in the following fields:

- planning
- commonsense reasoning
- (bounded) model checking
- VLSI (wire routing)
- Semantic Web
- Information Extraction
- . . .

Solving methods

- "native" methods that work directly on the (grounded) logic program, namely SMODELS and DLV
- methods based on compilation into a SAT theory (SAT-based ASP), namely CMODELS2 and ASSAT

Relation between ASP and SAT procedure

In some recent work (Giunchiglia and Maratea, ICLP 2005), we have shown that, on a wide and well studied set of logic programs called "tight" (where there is a 1 to 1 correspondence between the AS of Π and the solutions of $Comp(\Pi)$), the main search procedures used by "native" and SAT-based systems for ASP are equivalent, i.e., that they explore search trees with the same branching nodes.

The result has been proved for CMODELS2 and SMODELS, and extends to ASSAT and to DLV (work in progress).

Contribution of the work

In this work, we focus on the experimental evaluation of different search strategies, heuristics and their combinations that have been shown to be effective in the SAT community, in ASP systems.

Previous work (Faber, Leone and Pfeifer, IJCAI 2001; Faber and Ricca, LPNMR 2005), mostly considered and evaluated only one technique (the heuristic).

We used CMODELS2 as common reasoning platform, because it is SAT-based, strengthening in this way the relation between ASP and SAT, its back end SAT solver SIMO already incorporates various strategies and heuristics, and because it has a number of advantage w.r.t. ASSAT (Giunchiglia, Lierler and Maratea, AAAI 2004).

Results would extend (at least for the tight programs) to ASSAT and SMODELS (and DLV) if enhanced with corresponding techniques.

Agenda

- Review of the SAT-based algorithm
- Experimental analysis with several search strategies and heuristics
- Conclusions

CMODELS2 decision procedure

function $\textsc{Cmodels2}(\Pi)$ **return** $\textsc{dll}(\underline{\mathrm{CNF}(\mathit{Comp}(\Pi))}, \emptyset)$;
function $\textsc{dll}(\Gamma, S)$
  if $\Gamma = \emptyset$ **then return** $\underline{\mathit{test}(S, \Pi)}$;
  if $\emptyset \in \Gamma$ **then return** *False*;
  if $\{l\} \in \Gamma$ **then return** $\textsc{dll}(\mathit{assign}(l, \Gamma), S \cup \{l\})$;
  $A := \mathit{Heuristic}(\varphi)$;
  return $\textsc{dll}(\mathit{assign}(A, \Gamma), S \cup \{A\})$ **or**
    $\textsc{dll}(\mathit{assign}(\neg A, \Gamma), S \cup \{\neg A\})$

$\mathit{test}(S, \Pi)$ returns *True* if $S \cap P$ is an answer set of Π, and *False*, otherwise.

CMODELS2: Discussion

1. CMODELS2(Π) returns *True* iff Π has an answer set
2. CMODELS2(Π) can be easily modified in order to compute all the answer sets of a program Π
3. $\mathit{test}(S, \Pi)$ can fail because of "loops" in the logic program. Ex. Π: $p \leftarrow p$, $\mathit{Comp}(\Pi)$ is $p \equiv p$
4. CMODELS2 works in polynomial-space

Search strategies and heuristics

- Look-ahead: unit-propagation, based on lazy data structures; (denoted with "u");
- Look-ahead: unit-propagation+failed-literal detection. (denoted with "f").
- Look-back: basic backtracking; (denoted with "b");
- Look-back: backtracking+backjumping+learning. (denoted with "l").
- Heur Static: based on the order induced by the appearance in the SAT formula. (denoted with "s").
- Heur VSIDS: based on the information extracted from the optimized look-back phase of the search. (denoted with "v").
- Heur Unit: based on the information extracted from the failed-literal detection technique. (denoted with "u").
- Heur Unit with pool: Unit heuristic restricted to a subset of the open (not yet assigned) atoms. (denoted with "p").

Tight logic programs (I): Static heuristic

	PB	# VAR	uls	ubs	fls	fbs
1	4	300	TIME	TIME	**230.86**	338.05
2	5.5	300	TIME	TIME	**478.46**	TIME
3	6	300	371.28	TIME	120.02	**84.16**
4	bw-large.d9	9956	**0.9**	2497.02	2.68	2.62
5	bw-large.e10	13482	**1.61**	TIME	5.28	19.52
6	queens21	925	**0.20**	0.23	0.36	0.38
7	queens24	1201	**0.46**	1.14	0.67	0.74
8	queens50	5101	**3.67**	TIME	12.41	TIME
9	dp-12.fsa-i-b9	1186	**12.51**	2651.28	20.30	TIME
10	key-2-i-b29	3199	157.29	TIME	**111.61**	293.37
11	mmgt-3.fsa-i-b10	1933	TIME	TIME	**1570.27**	3241.45
12	mmgt-4.fsa-s-b8	1586	**1004.36**	TIME	1054.06	TIME
13	p1000	14955	**7.69**	TIME	377.02	TIME
14	p3000	44961	**178.26**	TIME	TIME	TIME
15	p6000	89951	**1275.62**	TIME	TIME	TIME

Table: Problems (1-3) are randomly generated; (4-5) are blocks-world; (6-8) are queens; (9-12) are bounded model checking; (13-15) are 4-colorability.

Tight logic programs (II)

	PB	# VAR	ulv	flv	flu	fbu	ulp	ubp
16	4	300	**0.41**	0.52	0.85	0.66	21.79	3.01
17	4.5	300	TIME	TIME	81.92	**22.53**	TIME	54.7
18	5	300	448.21	485.36	8.27	**4.72**	452.75	14.35
19	bw-large.d9	9956	1.02	5.84	2.69	2.75	**1.01**	TIME
20	bw-large.e10	13482	**1.29**	7.51	5.03	4.95	1.55	TIME
21	queens21	925	786.14	1864.49	384.87	47.33	**0.24**	**0.24**
22	queens24	1201	TIME	TIME	TIME	368.76	**0.28**	0.29
23	queens50	5101	TIME	TIME	TIME	TIME	347.98	**43.16**
24	dp-12.fsa-i-b9	1186	**223.93**	383.66	353.53	TIME	2910.96	1051.17
25	key-2-i-b29	3199	415.54	204.87	**44.14**	589.45	1329.53	TIME
26	mmgt-3.fsa-i-b10	1933	16.23	32.23	26.71	16.55	**6.19**	372.54
27	mmgt-4.fsa-s-b8	1586	17.02	27.59	421.30	327.55	**13.79**	2492.62
28	p1000	14955	**0.48**	37.86	15.41	15.23	3.69	TIME
29	p3000	44961	**8.86**	369.27	144.12	142.83	223.62	TIME
30	p6000	89951	**99.50**	TIME	583.55	578.98	2549.50	TIME

Table: Performances on tight programs. The problems are the same as in Table 1.

Non-tight logic programs (I): Static heuristic

	PB	# VAR	uls	ubs	fls	fbs
31	3	300	9.75	31.63	4.69	**4.4**
32	7.5	300	TIME	TIME	TIME	**567.78**
33	8	300	544.83	TIME	199.05	**178.98**
34	bw-basic-P4-i	5301	**2.08**	43.19	4.07	6.91
35	bw-basic-P4-i-1	4760	**1.73**	15.55	2.54	2.57
36	bw-basic-P4-i+1	5842	**2.29**	47.09	5.04	8.17
37	np60c	10742	**6.8**	TIME	125.83	TIME
38	np70c	14632	**12.34**	TIME	326.34	TIME
39	np80c	19122	**19.89**	TIME	745.26	TIME

Table: Problems (31-33), are randomly generated; (34-36) are blocks-world; (37-39) are Hamiltonian Circuit on complete graphs.

Non-tight logic programs (II)

	PB	# VAR	ulv	flv	flu	fbu	ulp	ubp
40	4	300	265.43	218.48	41.97	**31.05**	77.41	123.31
41	5	300	TIME	TIME	136.67	**99.75**	439.71	323.15
42	6	300	TIME	TIME	107.34	**65.83**	591.3	337.45
43	bw-b-P4-i	5301	**2.16**	15.54	6.07	5.79	2.54	79.64
44	bw-b-P4-i-1	4760	**1.64**	4.92	2.47	2.44	1.86	13.44
45	bw-b-P4-i+1	5842	2.49	24.27	22.01	19.71	**2.41**	11.60
46	np60c	10742	**2.83**	1611.32	44.12	44.12	4.77	597.82
47	np70c	14632	**4.69**	TIME	97.44	97.89	5.91	TIME
48	np80c	19122	**6.91**	TIME	192.29	196.32	12.88	TIME

Table: Performances on non-tight programs. The problems presented are the same as in Table 3.

Conclusions

- Learning is usually effective on real-world programs while failed-literal is effective on randomly generated programs
- results extend to non-tight programs (at least on the experimental side)
- an ulp-based solver is, at the moment, the most effective overall option
- we have shed light on future development: As soon as the number of variables in the challenges benchmarks will increase, for real-world problems we expect that ulv-based solvers, leaders in the SAT community, will become leaders also in ASP