An Experimental Study of Search Strategies and Heuristics in Answer Set Programming

E. Giunchiglia and M. Maratea

STAR-Lab
University of Genova, Italy

Bath, ASP05

Relation between ASP and SAT procedures

In some recent work (Giunchiglia and Maratea, to be presented at ICLP 2005),
we have shown that, on "tight" non-disjunctive logic programs, 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 CMODELS and SMODELS, and extends to ASSAT, SMODELS-CC and DLV (work in progress).

Consequences of the result on tight programs

Thanks to the previous theoretical result, we are able to establish new complexity results for SMODELS (and CMODELS)

1. Consider a random k-CNF formula Γ with n atoms and m clauses. With probability tending to one as n tends to infinity, the complexity of SMODELS and CMODELS on *sat2tlp*(Γ) is exponential in n if the *density* d=m/n $\geq 0.7 \times 2^k$.
2. The complexity of SMODELS and CMODELS on *sat2tlp*(PHP^n_{n-1}) is exponential in n.
3. In SMODELS, deciding the optimal literal to branch on is both NP-hard and co-NP hard, and in PSPACE for tight programs.

This complexity results extend to ASSAT and SMODELS-CC as well.

Consequences on non-tight programs

A modification in the definition of *sat2tlp*(Γ) in 2., such that the resulting logic program is non-tight, is enough to prove that SMODELS and CMODELS are not equivalent on non-tight programs.

For such a modification, SMODELS performs *exponentially* better than CMODELS.

Motivations

Given the strong relation established, one can expect that search strategies and heuristics that have been demonstrated to be effective for SAT, to be effective also in ASP, at least on tight programs.

Contribution of this 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 2001; Faber and Ricca 2005), mostly considered and evaluated one technique (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
- it has a number of advantage w.r.t. ASSAT, see (Giunchiglia et al. AAAI 2004).

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

Search strategies and heuristics

1. Look-ahead unit-propagation, based on lazy data structures ("u");
2. Look-ahead unit-propagation+failed-literal detection. ("f").
3. Look-back basic backtracking ("b");
4. Look-back backtracking+backjumping+learning. ("l").
5. Heuristic Static, based on the order induced by the appearance in the SAT formula ("s");
6. Heuristic VSIDS, based on the information extracted from the optimized look-back phase of the search ("v");
7. Heuristic Unit, based on the information extracted from the failed-literal detection technique ("u");
8. Heuristic Unit with pool, Unit heuristic restricted to a subset of the open (not yet assigned) atoms ("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-basic-P4-i	5301	**2.16**	15.54	6.07	5.79	2.54	79.64
44	bw-basic-P4-i-1	4760	**1.64**	4.92	2.47	2.44	1.86	13.44
45	bw-basic-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.

Summing up...

- 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