Propositional satisfiability (SAT), SAT-based ASP and relation between ASP and SAT

Marco Maratea
(j.w.w. **Enrico Giunchiglia**)

Dipartimento di Matematica
Università della Calabria

DEIS, UNICAL, 14/06/2006

Motivation

1. Propositional satisfiability (SAT) is one of the most studied fields in AI and CS
2. Very efficient and specialized SAT procedures exist
   - $\Rightarrow$ use SAT solvers for deciding more expressive logics and formalisms ...
   - $\Rightarrow$ ... reusing most of the work and knowledge available in SAT

SAT: The problem

A *literal* l is a proposition (variable/atom) p or its negation $\neg p$.
Given the literals $l_1, \ldots, l_k$, a *clause* is $l_1 \vee \cdots \vee l_k$.
Given the clauses $c_1, \ldots, c_m$, a Conjunctive Normal Form (CNF) formula is $c_1 \wedge \cdots \wedge c_m$.
An *assignment*, or *valuation* v, is a partial function from the propositions to $\{\text{TRUE},\text{FALSE}\}$.
We can extend the definition of v in the natural way to assign truth values to literals, clauses and formulas.

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

> Does there exist an assignment v to the propositions in Γ such that Γ is true?

SAT: Examples

1. $\varphi := \{p, p \vee \neg q, \neg r\}$ has the satisfying assignments
   - $\{p := \text{TRUE}, \quad q := \text{TRUE}, \quad r := \text{FALSE}\}$
   - $\{p := \text{TRUE}, \quad q := \text{FALSE}, \quad r := \text{FALSE}\}$

2. $\varphi := \{\neg p, p \vee \neg q, r \vee \neg p, q\}$ has no satisfying assignments because the clause $\{p \vee \neg q\}$ can not be satisfied.

SAT: Solving methods

- Resolution algorithm
- Local search algorithms
- (Ordered) Binary Decision Diagrams (OBDDs) (Bryant 1992)
- Davis-Logemann-Loveland (DLL) algorithm

Agenda

- DLL algorithm
- SAT/DLL-based approaches to ASP
- Experiments with ASP solvers
- Relation between ASP and SAT procedures
- Further experiments

DLL algorithm

function $\text{DLL-REC}(\Gamma, S)$
  $\langle \Gamma, S \rangle := \textit{unit-propagate}(\Gamma, S)$;
  if $(\emptyset \in \Gamma)$ **return** FALSE;
  if $(\Gamma = \emptyset)$ **return** TRUE;
  $A := \textit{ChooseAtom}(S)$;
  return $\text{DLL-REC}(\textit{s-assign}(A, \Gamma)), S \cup \{A\})$ **or**
    $\text{DLL-REC}(\textit{s-assign}(\overline{A}, \Gamma)), S \cup \{\overline{A}\})$;

function $\textit{unit-propagate}(\Gamma, S)$
  if $(\{l\} \in \Gamma)$ **return** $\textit{unit-propagate}(\textit{s-assign}(l, \Gamma), S \cup \{l\})$;
  return $\langle \Gamma, S \rangle$;

$\textit{s-assign}(l, \Gamma)$ deletes all the clauses containing l, and all the occurrences of $\overline{l}$.

Introduction to ASP

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

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

Let P be the set of atoms in Π, $A_0 \in P \cup \{\bot\}$, $\{A_1, \ldots, A_n\} \subseteq P$. A_0 is the *head*.

Comp(Π) (Clark 1978) 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.

For the class of *tight* logic programs (Fages 1994; Erdem and Lifschitz 2003), there is 1-1 correspondence between the ASP solutions and the models of *Comp*(Π).

SAT-based ASP: Previous approaches (I)

CMODELS algorithm (focus on finding one answer set)

1. Computes $\Gamma = Comp(\Pi)$. (and converts it into a set of clauses)
2. Checks if Γ is tight.
3. If it is, finds a model X of Γ by a SAT solver. If such a model exists returns TRUE, otherwise FALSE.

SAT-based ASP: Previous approaches (II)

ASSAT algorithm (Lin and Zhao 2002,2004)

1. Computes $\Gamma = Comp(\Pi)$. (and converts it into a set of clauses)
2. Finds a model X of Γ by a SAT solver. If no such a model exists, return FALSE.
3. Checks if X is an answer set: If X is an answer set, then returns TRUE. Otherwise
   a) finds (at least) one “loop formula” which is not satisfied by X, and adds it to Γ; and
   b) goes back to step 2.

The foundation of the algorithm is in the following theorem:

Theorem

Let LF(Π) be the set of loop formulas associated with the loops of Π. X is an answer set iff is a model of $Comp(\Pi) \cup$ LF(Π).

ASSAT disadvantages

- It is not guaranteed to work in polynomial space.
- Some computation can be repeated. (several times)
- It introduces new variables, other than the ones needed by the clause-form transformation

Besides these weakness, ASSAT showed to be very competitive wrt state-of-the-art systems like SMODELS and DLV.

But (much) better could be done not considering the SAT solver as a “black-box”.

CMODELS2: DLL-based decision procedure for ASP

function CMODELS2(Π) **return** DLL-REC(*lp2sat*(Π),$\emptyset$,Π);

function DLL-REC(Γ,S,Π)
  $\langle \Gamma, S \rangle := \textit{unit-propagate}(\Gamma, S)$;
  if $(\emptyset \in \Gamma)$ **return** FALSE;
  if $(\Gamma = \emptyset)$ **return** *test*(S,Π);
  $A := \textit{ChooseAtom}(S)$;
  return DLL-REC($\textit{s-assign}(A, \Gamma)), S \cup \{A\}$) **or**
      DLL-REC($\textit{s-assign}(\overline{A}, \Gamma)), S \cup \{\overline{A}\}$);

function *unit-propagate*(Γ,S)
  if $(\{l\} \in \Gamma)$ **return** $\textit{unit-propagate}(\textit{s-assign}(l, \Gamma), S \cup \{l\})$;
  return $\langle \Gamma, S \rangle$;

CMODELS2 employs the SAT solvers SIMO, a ZCHAFF-like solver.
$\textit{test}(S, \Pi)$ returns TRUE if $S \cap P$ is an answer set of Π, and FALSE, otherwise.

Extension to non-basic rules

CMODELS2 can work with other types of rules other than the basic ones showed before, namely:

- choice rules,
  $\{A_0, \dots, A_k\} \leftarrow A_{k+1}, \dots, A_m, not\ A_{m+1}, \dots, not\ A_n$
- cardinality and weight constraint rules

  $A_0 \leftarrow L\{A_1 = w_1, \dots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \dots, not\ A_n = w_n\}U$

All these rules, together with the basics, can be translated into *basic nested* rules

$$A_0 \leftarrow A_1, \dots, A_m, not\ A_{m+1}, \dots, not\ A_k, not\ not\ A_{k+1}, \dots, not\ not\ A_n.$$

A choice rule $\{A\} \leftarrow .$ is translated in $A \leftarrow not\ not\ A$, while weight constraint rules are translated using the method presented in (Ferraris and Lifschitz, TPLP 2005).
For a basic nested program Π, $Comp(\Pi)$ is defined as well.

CMODELS2: Discussion

1. CMODELS2(Π) returns TRUE iff Π has an answer set
2. CMODELS2 works in polynomial space
3. CMODELS2(Π) can be modified in order to compute all the answer sets of a program Π
4. *test*(S, Π) can fail because of "loops" in Π
5. Most state-of-the-art SAT solvers are a (non-recursive) implementation of DLL
6. Most state-of-the-art SAT solvers are based on "learning" in order to backjump irrelevant nodes while backtracking and avoid the exploration of useless parts of the search tree

CMODELS2: Computing reasons

If SAT solvers are based on learning

1. Learning procedures require $test(S, \Pi)$ to return a $S' \subseteq S$ such that for each S'' entailing $Comp(\Pi)$ and with $S' \subseteq S''$, $S'' \cap P$ is ensured not to be an AS of Π
2. One such set is S, but it is important that S be as small as possible:
   - $\Rightarrow$ one possibility it to return $S \cap P$, or (better)
   - $\Rightarrow$ we can compute a subset of S which falsifies one of the loop formulas in Π

CMODELS2: Advantages

With respect to ASSAT, CMODELS2 has a number of advantages, other than points 2. and 3. in the discussion slide

- it works with basic and non-basic rules
- no computation is ever repeated
- it does not introduce extra variables (except the ones needed by the clause form transformation)

With respect to SMODELS and DLV, CMODELS2 has the advantage of being SAT-based, and thus it can leverage on the great amount of work done in SAT

Experimental results: Tight programs

	SMODELS	SMODELS-CC	ASSAT	DLV	CMODELS2
4c1000	22.28	4.95	<u>0.6</u>		**0.48**
4c3000	202.84	1143.13	**2.19**		8.86
4c6000	856.13	TIME	**14.85**		99.50
schur.4-43	**0.43**	0.95	<u>0.67</u>	590.57	1.4
schur.4-44	**0.44**	91.25	1.07	TIME	5.97
schur.4-45	571.17	1110.68	<u>434.93</u>	TIME	**229.04**
15puz.18	17.55	6.94	<u>1.06</u>	141.68	**0.98**
15puz.19	20.94	7.14	3.61	208.41	**1.35**
15puz.20	70.27	8.22	4.59	TIME	**1.28**
pige.9.10	44.77	65.91	**1.1**		<u>1.26</u>
pige.10.11	484.63	1029.38	<u>23.83</u>		**12.41**
pige.51.50	106.79	24.29	<u>2.49</u>		**1.63**

Table: 4c* = 4 coloring; schur* = schur numbers; 15puz* = puzzle; and pige* = pigeons programs.

Experimental results: Blocks world

		Standard programs				Extended programs		
#b	#s	SM	SMCC	ASSAT	CM2	SM	SMCC	CM2
8	i-1	7.48	7.17	0.86	**0.49**	0.53	0.66	**0.15**
11	i-1	36.18	35.53	3.15	**1.64**	1.6	1.96	**0.39**
8	i	17.35	9.3	0.98	**0.63**	0.76	0.8	**0.22**
11	i	37.71	43.9	3.59	**2.16**	1.87	2.57	**0.52**
8	i+1	12.08	15.17	**1.09**	1.34	1.8	1.05	**0.68**
11	i+1	54.3	62.39	3.9	**2.49**	2.5	4.12	**0.6**

Table: Blocks world: “#b” is the number of blocks. “#s” is the number of steps.

Experimental results: H.C. complete graphs

	Standard programs					Extended programs		
	SM	SMCC	ASSAT	DLV	CM2	SM	SMCC	CM2
np30c	4.08	2.18	1.61	9.55	**0.35**	0.26	0.68	**0.17**
np40c	22.05	6.57	70.97	44.7	**0.85**	0.75	1.16	**0.66**
np50c	84.49	15.3	24.17	142.55	**1.66**	2.3	**1.95**	2.21
np60c	242.61	30.81	84.87	361.8	**2.83**	7.05	3.82	**3.55**
np70c	557.08	55.31	520.8	798.96	**4.69**	15.67	**5.92**	10.54
np80c	1001.88	90.59	53.25	1587.6	**7.2**	32.29	**9.01**	15.05
np90c	2064.61	144.72	1416.24	2807.84	**10.42**	53.21	**14.13**	32.19
np100c	3573.19	215.37	TIME	TIME	**14.23**	83.11	**14.95**	34.18

Table: Complete graphs. npXc corresponds to a graph with "X" nodes.

Experimental results: Formal Verification problems

	SMODELS	SMODELS-CC	ASSAT	DLV	CMODELS2
mutex4	14.14	5.35	0.54	367.89	**0.46**
phi4	**0.18**	0.55	1.96	0.83	179.71
mutex2	0.28	0.3	2.6		**0.15**
mutex3	163.94	**110.27**	MEM		TIME
phi3	3.23	3.04	53.28		**1.43**

Table: Checking requirements in a deterministic automaton. (Heljanko and Stefanescu 2003)

Experimental results: BMC problems

BMC	SMODELS	SMODELS-CC	CMODELS2
dp-10.i-O2-b12	132.72	**2.25**	488.76
dp-10.s-O2-b9	9.75	**3.11**	6.38
dp-12.s-O2-b10	296.45	**1.1**	53.2
dp-8.i-O2-b10	**1.76**	<u>2.42</u>	12.28
dp-8.s-O2-b8	0.73	**0.14**	0.47

Table: Bounded Model Checking Problems. (Heljanko and Niemela 2003)

Enhancements to CMODELS

From the point of view of search

1. Introduce new SAT techniques (see next part of the talk!)
2. Design specialized heuristic (see next slide!)
3. Integrate a new SAT solver
4. Loop (formulas) are “too generous” (elementary loops/sets)

From the point of view of expressivity, CMODELS3

1. Extension of CMODELS2 that allows for (non-nested) disjunctive rules, choice and weight constraints rules
2. *test*() is a co-NP check: It uses (another) SAT solver for it
3. Interesting preliminary results, more has to be done

"Heuristic false" (hf) tie-breaking heuristic

	Extended programs			
	SMODELS	SMODELS-CC	CMODELS2	CMODELS2 hf
np30c	0.26	0.68	0.17	**0.16**
np40c	0.75	1.16	0.66	**0.45**
np50c	2.3	1.95	2.21	**0.91**
np60c	7.05	3.82	3.55	**1.74**
np70c	15.67	5.92	10.54	**2.96**
np80c	32.29	9.01	15.05	**4.73**
np90c	53.21	14.13	32.19	**7.61**
np100c	83.11	14.95	34.18	**10.79**

	SMODELS	SMODELS-CC	ASSAT	DLV	CMODELS2	CM2 hf
mutex4	14.14	5.35	0.54	367.89	0.46	**0.34**
phi4	**0.18**	0.55	1.96	0.83	179.71	TIME
mutex2	0.28	0.3	2.6		0.15	**0.07**
mutex3	163.94	110.27	MEM		TIME	**11.82**
phi3	3.23	3.04	53.28		**1.43**	1.94

SAT-based: Other applications

The SAT/DLL-based approach has been used (in our group) to develop decision procedures for

- Separation Logic, a decidable quantifier-free fragment of the first order logic involving propositional logic and linear arithmetic, with applications in FV and scheduling (TSAT++)

Example

$\text{START} \wedge ((e_i - s_i \leq 10) \vee (s_j - e_i \leq 0))$

- optimization problem related to SAT (namely Max-SAT, Min-ONE) with main application in planning (OPTSAT)

and

- QSAT, or QBF, (QuBE++)
- conformant planning (CPlan)

On the relation between ASP and SAT procedures: Motivation

- The relation between ASP and SAT has been at the center of several papers, especially in the last years.
- This is confirmed by the upcoming new (ICLP-)workshop Lash06: Search and Logic: Answer Set Programming and SAT.
- Despite state-of-the-art ASP solvers are apparently quite different,
- the main search procedures used by ASP solvers (i.e., "native" and SAT-based) have been advocated "similar" in many works. But this has never been formally stated before.

On the relation between AS and SAT procedures: Goal

We study the computational properties of ASP systems, in order to formally characterize under which conditions different systems have same behavior.

We begin our study with SMODELS and CMODELS2, and then we see how the results extend to other systems like DLV, SMODELS-CC and ASSAT.

The main focus of this work is on *tight* programs using basic rules, where we will establish a strong relation between SMODELS and CMODELS2 procedures.

We will use the result both on the theoretical side (in order to show new complexity results for SMODELS) and on the experimental side (for evaluating efficient strategies and heuristics coming from SAT, in ASP systems).

SMODELS procedure (I)

function SMODELS(Π) **return** SMODELS-REC(Π, $\{\top\}$);

function SMODELS-REC(Π,S)
    $\langle \Pi, S \rangle := expand(\Pi, S)$;
    if $(\{l, not\ l\} \subseteq S)$ **return** FALSE;
    if $(\{A : A \in P, \{A, not\ A\} \cap S \neq \emptyset\} = P)$ **return** TRUE;
    $A := ChooseAtom(S)$;
    return SMODELS-REC($p\text{-}elim(A, \Pi)), S \cup \{A\}$) **or**
            SMODELS-REC($p\text{-}elim(not\ A, \Pi)), S \cup \{not\ A\}$);

function $expand(\Pi,S)$
    $S' := S$;
    $S := AtLeast(\Pi, S)$;
    $\Pi := p\text{-}elim(S, \Pi)$;
    $S := S \cup \{not\ A : A \in P, A \notin AtMost(\Pi^{\emptyset}, S)\}$;
    $\Pi := p\text{-}elim(S, \Pi)$;
    if $(S \neq S')$ **return** $expand(\Pi,S)$;
    return $\langle \Pi, S \rangle$;

SMODELS procedure (II)

function $AtLeast(\Pi, S)$
  if $(r \in \Pi$ **and** $body(r) = \emptyset$ **and** $head(r) \notin S)$
    return $AtLeast(p\text{-}elim(head(r), \Pi), S \cup \{head(r)\});$
  if $(\{A, not\ A\} \cap S = \emptyset$ **and** $\nexists r \in \Pi : head(r) = A)$
    return $AtLeast(p\text{-}elim(not\ A, \Pi), S \cup \{not\ A\});$
  if $(r \in \Pi$ **and** $head(r) \in S$ **and** $body(r) \neq \emptyset$ **and**
    $\nexists r' \in \Pi, r' \neq r : head(r') = head(r))$
    return $AtLeast(p\text{-}elim(body(r), \Pi), S \cup body(r));$
  if $(r \in \Pi$ **and** $not\ head(r) \in S$ **and** $body(r) = \{l\})$
    return $AtLeast(p\text{-}elim(not\ l, \Pi)), S \cup \{not\ l\});$
  return S;

function $AtMost(\Pi, S)$
  if $(r \in \Pi$ **and** $body(r) = \emptyset$ **and** $head(r) \notin S)$
    return $AtMost(p\text{-}elim(head(r), \Pi), S \cup \{head(r)\});$
  return S;

From a logic program to a set of clauses

We have defined $lp2sat(\Pi)$ to be the set of clauses corresponding to $Comp(\Pi)$. More precisely, if A_0 is an atom, the *translation of* Π *relative to* A_0, denoted with $lp2sat(\Pi, A_0)$, consists of

1. for each rule $r \in \Pi$ of the form (1) and whose head is A_0, the clauses:

$$\{A_0, \overline{n}_r\}, \{n_r, \overline{A}_1, \ldots, \overline{A}_m, A_{m+1}, \ldots, A_n\},$$
$$\{\overline{n}_r, A_1\}, \ldots, \{\overline{n}_r, A_m\}, \{\overline{n}_r, \overline{A}_{m+1}\}, \ldots, \{\overline{n}_r, \overline{A}_n\},$$

   where n_r is a newly introduced atom, and

2. the clause $\{\overline{A}_0, n_{r_1}, \ldots, n_{r_q}\}$ where $n_{r_1}, \ldots, n_{r_q}$ $(q \geq 0)$ are the new symbols introduced in the previous step.

The *translation of* Π *relative to* $\perp$, denoted with $lp2sat(\Pi, \perp)$, consists of a clause $\{\overline{A}_1, \ldots, \overline{A}_m, A_{m+1}, \ldots, A_n\}$, one for each rule in Π of the form (1) with head $\perp$. Finally, the *translation of* Π, denoted with $lp2sat(\Pi)$, is $\cup_{p \in P \cup \{\perp\}} lp2sat(\Pi, p)$.

From a set of clauses to a logic program

If C is a clause $\{l_1, \ldots, l_l\}$ ($l \geq 0$) we define *sat2tlp*(C) to be the rule

$$\perp \leftarrow not\ l_1, \ldots, not\ l_l.$$

Then, if Γ is a formula, the *translation of* Γ, denoted with *sat2tlp*(Γ), is

$$\cup_{C \in \Gamma} sat2tlp(C) \cup \cup_{p \in P} \{p \leftarrow not\ p', p' \leftarrow not\ p\}$$

where, for each atom $p \in P$, p' is a new atom associated to p.

Relating SMODELS and CMODELS2

Our goal is to prove that the computations of SMODELS and CMODELS2 are highly related if Π is tight. We establish this comparing the search trees of SMODELS-REC$(\Pi, \{\top\})$ and DLL-REC$(lp2sat(\Pi), \emptyset)$.

We say that a set of literals S is a *branching node* of SMODELS(Π) if there is a call to SMODELS-REC(Π', S), following the invocation of SMODELS(Π). Similar considerations are made for CMODELS2. If *proc* is SMODELS(Π) or CMODELS2(Π), we define

$$Br(proc) = \{S \cap (P \cup \overline{P}) : S \text{ is a branching node of } proc\}.$$

We say that SMODELS(Π) and CMODELS2(Π) are *equivalent* if $Br(\text{SMODELS}(\Pi)) = Br(\text{CMODELS2}(\Pi))$.

Theorem

For each tight program Π, SMODELS(Π) and CMODELS2(Π) are equivalent.

New results for SMODELS: Pigeonhole principle

The *complexity of a procedure proc on a program* Π is the smallest N such that $|Br(proc)| = N$.
Consider the formula PHP_n^m where n, m are two natural numbers, and consisting of the clauses

$$\begin{array}{ll} \{A_{i,1}, A_{i,2}, \ldots, A_{i,n}\} & (i \leq m), \\ \{\overline{A}_{i,k}, \overline{A}_{j,k}\} & (i, j \leq m, k \leq n, i \neq j). \end{array}$$

The formulas PHP_n^m are from (Haken 1985) and encode the pigeonhole principle. If $n < m$, PHP_n^m are unsatisfiable and it is well known that any procedure based on resolution (like DLL-REC) has an exponential behavior on these formulas.

Corollary

The complexity of SMODELS *and* CMODELS2 *on sat2tlp*(PHP_{n-1}^n) *is exponential in n.*

The result extends to CMODELS2 because it is based on DLL-REC. For SMODELS, it relies on the fact that *sat2tlp*(PHP_{n-1}^n) is tight, and thus SMODELS and CMODELS2 are equivalent.

New results for SMODELS: Randomly generated k-CNF formulas

A formula Γ is a *k-CNF* if each clause in Γ consists of k literals. The *random family of k-CNF formulas* is a k-CNF whose clauses have been randomly selected with uniform distribution among all the clauses C of k literals and such that, for each two distinct literals l and l' in C, $\bar{l} \neq l'$.

Corollary

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* CMODELS2 *on sat2tlp(Γ) is exponential in n if the density $d = m/n \geq 0.7 \times 2^k$.*

This result follows from (Chvátal and Szemerédi 1988), and again from the fact that *sat2tlp*(Γ) is tight on the random family, from the fact that CMODELS2 is based on DLL-REC and our equivalence result on tight programs.

New results for SMODELS: Deciding the best literal

We define a literal l to be *optimal for a program* Π if there exists a minimal search tree of SMODELS(Π) whose root is labeled with l. The following result echoes the one by (Liberatore 2000) for DLL-REC.

Corollary

In SMODELS, *deciding the optimal literal to branch on is both NP-hard and co-NP hard, and in PSPACE for tight programs.*

There are many other results holding for DLL-REC that can be lifted to SMODELS, including (Monasson 2004) and (Achlioptas et al. 2001) for average complexity of coloring randomly generated graphs and for exponential lower bounds on random 3-CNF formulas also below the satisfiability threshold.

SMODELS and CMODELS2 are not equivalent on non-tight programs

Consider again the pigeonhole formulas. They give us the opportunity to define a class of formulas that are exponentially hard for CMODELS2 but easy for SMODELS.

For each formula Γ, defines $sat2nlp(\Gamma)$ to be the program $\cup_{C \in \Gamma} sat2tlp(C) \cup \cup_{p \in P} \{p \leftarrow p\}$.

Corollary

The complexity of SMODELS *and* CMODELS2 *on* $sat2nlp(PHP^n_{n-1})$ *is* 0 *and exponential in* n *respectively.*

In this case, $sat2nlp(PHP^n_{n-1})$ is non-tight, and SMODELS can determine the non existence of answer sets without branching mainly thanks to the procedure *AtMost*.

The above results can be easily generalized to any formula Γ which is known to be exponentially hard for DLL-REC.

Extending the results to other systems

ASSAT is different from CMODELS2 only on non-tight programs, assuming that Γ is computed as *lp2sat*(Π).

SMODELS-CC is SMODELS enhanced with "clause-learning" look-back strategies.

Results in (Haken 1985) and (Chvátal and Szemerédi 1988) hold for any proof systems based on resolution. Enhancing SMODELS and CMODELS2 with "learning" look-back strategies does not lower the exponential complexity.

Thus, the related corollaries hold also for SMODELS-CC and ASSAT.

DLV core algorithm is similar to the one of SMODELS. In particular, the rules used by *AtLeast* to extend the assignment S are very similar to those used by the DLV procedure *DetCons*. (see (Faber 2002), pagg. 41-44.)

We (Enrico, Nicola and I) are working on the comparison between DLV algorithm, DLL-REC (thus CMODELS) and SMODELS.

Experimental analysis: Assessment (I)

Given the above results, one expects that the combinations of reasoning strategies that currently dominate in SAT, are also bound to dominate in ASP, at least on tight logic programs.

We show experimentally, on a wide set of currently challenges benchmarks, that this is the case (to certain degrees), and results extend (on the experimental side) to non-tight programs.

We have used our solver, CMODELS2, because it is SAT-based and thus strengths the relation between SAT and ASP, and also

- its front-end is LPARSE (Simons 2000), a widely used grounder for logic programs;
- its back-end solver already incorporates (lazy) data structures for fast unit propagation as well as some state-of-the-art strategies and heuristics evaluated in this work; and
- can be also run on non-tight programs.

Experimental analysis: Assessment (II)

We have further extended CMODELS2 with a variety a look-ahead, look-back strategies and heuristics coming from the SAT community.

- Look-ahead: basic unit-propagation (u), unit-propagation+failed-literal (f) (Freeman 1995)
- Look-back: basic backtracking (b), backtracking+backjumping+learning (l) (Sakallah and Silva 1996; Bayardo and Schrag 1997; Zhang et. al 2001)
- Heuristic: VSIDS (v) (Moskewicz et al. 2001), Unit-based (u) (Li and Anbulagan 1997)

We focus on 4 combination of strategies built out of them: ulv, flv, flu and fbu.
Performing the experiment on a unique platform is of fundamental importance, otherwise results can be biased by implementation issues.
Given the established "equivalence", results would extend to SMODELS (and to the other systems, according to the considerations made) if enhanced with corresponding techniques (at least on tight programs).

Experimental analysis: Small, random programs

	PB	# VAR	ulv	flv	flu	fbu
1	4	300	**0.59**	0.8	1.5	1.37
2	4.5	300	TIME	TIME	115.29	**40.38**
3	5	300	456.22	538.89	17.64	**11.32**
4	5.5	300	72.83	53.26	4.42	**3.59**
5	6	300	24.73	21.89	1.83	**1.63**
6	4	300	265.43	218.48	41.97	**31.05**
7	4.5	300	TIME	TIME	190.73	**135.11**
8	5	300	TIME	TIME	136.67	**99.75**
9	5.5	300	TIME	TIME	129.29	**78.63**
10	6	300	TIME	TIME	107.34	**65.83**

Table: Performances on randomly generated logic programs. Problems (1)-(5) are tight programs being the translation of 3-SAT benchmarks. Problems (6)-(10) are randomly generated logic programs using Lin and Zhao's methodology.

Experimental analysis: Large programs

	PB	# VAR	ulv	flv	flu	fbu
11	bw*d9	9956+	**1.02**	5.84	2.69	2.75
12	bw*e9	12260	**0.98**	<u>1.91</u>	<u>1.92</u>	<u>1.93</u>
13	bw*e10	13482+	**1.29**	7.51	5.03	4.95
14	4c1000	14955+	**0.48**	37.86	15.41	15.23
15	4c3000	44961+	**8.86**	369.27	144.12	142.83
16	4c6000	89951+	**99.50**	TIME	583.55	578.98
17	np80c	19122+	**7.2**	TIME	195.08	190.49
18	np90c	24212+	**10.42**	TIME	364.54	357.92
19	np100c	29902+	**14.23**	TIME	610.2	608.96
20	np80c	19043+	**15.05**	1538.86	<u>23.63</u>	<u>25.94</u>
21	np90c	24123+	**32.19**	2918.82	<u>38.75</u>	<u>50.08</u>
22	np100c	29803+	**34.18**	TIME	<u>59.15</u>	<u>62.64</u>
23	mutex4	14698+	**0.46**	28.29	28.3	28.26
24	phi3	16930+	**1.43**	55.62	12.15	TIME

Table: (11)-(13) are blocks world; (14)-(16) are 4 coloring; (17)-(22) are HC on complete graphs; (23)-(24) are "verification" problems.

Experimental analysis: Non random, non large programs

	PB	# VAR	ulv	flv	flu	fbu
25	k*i*29	3199	415.54	204.87	**44.14**	589.45
26	k*s*29	3169	353.69	1028.77	**59.99**	TIME
27	q*i*17	2201	1539.96	<u>505.15</u>	**259.05**	816.26
28	e*3*i*15	7832+	479.28	TIME	<u>7.15</u>	**6.87**
29	e*4*i*13	6447	87.63	567.27	<u>20.02</u>	**19.41**
30	d*10*i*12	1488+	488.76	1212.89	**152.8**	TIME
31	d*10*s*9	1140+	**6.38**	19.31	87.64	TIME
32	d*12*s*10	1511+	**53.2**	165.9	733.9	TIME
33	d*8*i*10	1003+	12.28	25.03	**1.21**	11.86
34	d*8*s*8	819+	**0.47**	3.73	2.38	1221.53
35	schur.4-43	736+	<u>1.4</u>	2.07	**0.82**	<u>0.88</u>
36	schur.4-44	753+	<u>5.97</u>	**5.62**	92.63	43.01
37	schur.4-45	770	<u>229.04</u>	417.34	244.35	**116.51**
38	15puz.18	5945+	**0.98**	2.9	9.85	9.24
39	15puz.19	6258+	**1.35**	2.93	11.65	10.76
40	15puz.20	6571	**1.28**	10.22	64.54	82.68
41	pige.9.10	210	**1.26**	4.33	1259.84	32.06
42	pige.10.11	253	**12.41**	55.46	TIME	339.06
43	pige.51.50	5252+	**1.63**	221.33	6.85	7.26

Table: (25)-(34) are BMC; (35)-(37) are schur numbers; (38)-(40) are 15 puzzle; (41)-(43) are pigeons problems.

Possible enhancements

1. Design/Implement "better" look-ahead techniques (like DLV),
   - to "bound" the number of failed-literals
2. Design/Implement heuristic suited for random benchmarks
   - based on "backbones", but this point "calls" ...
   - ... for the research issue of generating random logic programs
3. Refine VSIDS heuristic
   - initialization and variables scoring

Main results

- the SAT-based approach used by CMODELS2 is competitive w.r.t. rival systems, at least on non-disjunctive case and when looking for one answer set
- ASP and SAT procedures have been demonstrated to be "equivalent" on tight programs; this lead to establish new, previously unknown results for SMODELS that can be extended to ASSAT and SMODELS-CC with the extents we have seen. Extending the results to DLV is work in progress
- a deep experimental investigation, motivated by the previous theoretical result, has shown how SAT techniques can be beneficial for ASP solvers, and has shed light on future directions for develop ASP systems

What am I doing @mat.unical?

1. Look-back (VSIDS-like) heuristic for DLV. Preliminary results
2. Extending the relation between SMODELS/CMODELS to DLV.

   (Expected) Results (to be proved ...)

   - DLV, SMODELS and CMODELS are equivalent on tight programs
   - DLV and SMODELS are equivalent on non-tight programs
   - DLV and SMODELS are not equivalent on non-tight programs if deprived of well-founded/atmost procedures.

Main references (I): ASP

- E. Giunchiglia, Yu. Lierler, M. Maratea - Answer Set Programming based on Propositional Satisfiability. Accepted to the Journal of Automated Reasoning (JAR).
- E. Giunchiglia, M. Maratea - On the relation between Answer Set and SAT procedures (or, between smodels and cmodels). In Proc. 21th International Conference on Logic Programming (ICLP 2005).
- E. Giunchiglia, Yu. Lierler and M. Maratea - SAT-based Answer Set Programming. In Proc. 19th American Association for Artificial Intelligence (AAAI 2004).
- Yu. Lierler and M. Maratea - Cmodels2: SAT-based Answer Set Solvers Extended to Non-tight Programs. In Proc. 7th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2004).

Main references (II): Others

- E. Giunchiglia, M. Maratea - Solving Optimization Problems with DLL. Accepted to the 17th European Conference on Artificial Intelligence (ECAI) 2006.
- M. Maratea - Efficient Decision Procedures for the Integration of Planning and Formal Verification in Advanced Systems. AI Communications. IOS Press. 2006.
- A. Armando, C. Castellini, E. Giunchiglia and M. Maratea - The SAT-based Approach to Separation Logic. Journal of Automated Reasoning (JAR). 2006.
- E. Giunchiglia, M. Maratea and A. Tacchella - (In)Effectiveness of Look-ahead Techniques in a Modern SAT solver. In Proc. 9th Int. Conference on Principle and Practice of CP (CP 2003).
- E. Giunchiglia, M. Maratea and A. Tacchella - Dependent and Independent Variables in Propositional Satisfiability. In Proc. 8th European Conference on Logics in AI (JELIA 2002).
- E. Giunchiglia, M. Maratea, A. Tacchella and D.Zambonin - Evaluating Search Heuristics and Optimization Techniques in Propositional Satisfiability. In Proc. 1st International Joint Conference on AR (IJCAR 2001).