Pbmodels — Software to Compute Stable Models by Pseudoboolean Solvers

Lengning Liu and Mirosław Truszczyński

Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046, USA

Abstract. We describe a new software, *pbmodels*, that uses pseudo-boolean constraint solvers (PB solvers) to compute stable models of logic programs with weight atoms. To this end, *pbmodels* converts ground logic programs to propositional theories with weight atoms so that stable models correspond to models. Our approach is similar to that used by *assat* and *cmodels*. However, unlike these two systems, $pbmodels$ does not compile the weight atoms away. Preliminary experimental results on the performance of $pbmodels$ are promising.

1 Introduction

We describe a system *pbmodels* to compute stable models of logic programs with weight atoms. We call such programs $smodels$ programs, as we adopt for them the semantics implemented in *smodels* [1]. The key idea behind *pbmodels* is to translate programs into propositional theories and use propositional satisfiability solvers. However, unlike existing systems *assat* [2] and *cmodels* [3], which first exploited this direction, we do not replace weight atoms with propositional formulas. Instead, we translate programs with weight atoms directly into theories in propositional logic *extended* with weight atoms, which we refer to as the logic PL^{wa} [4]. We then use existing solvers for such theories as computational back-end engines. We refer to solvers testing satisfiability of PL^{wa} theories as PB solvers. In some cases, prior to the use of a PB solver, additional simple transformations are needed to ensure the right input format.

Compiling away weight atoms may lead to significantly larger programs and theories. Currently most advanced translations result in the growth by a logarithmic factor in the case of cardinality atoms, and a polynomial factor in the case of general weight atoms. Moreover, for some solvers, especially those based on the local-search idea, the structure of the resulting theories makes them difficult to process. The growth in size and the additional structure, which result from compiling weight atoms away, often render solvers that require that step, such as *assat* [2] and *cmodels* [3], less effective. That motivates our work. *Pbmodels* is designed to compute models of *smodels* programs *without* replacing weight atoms by their pure propositional representations.

2 *Pbmodels* — the Algorithm

Our paper is based on theoretical results developed in [4] for an abstract setting of programs with monotone and convex constraints. The approach developed there specializes to $smodels$ programs. In particular, it yields the concepts of the program completion, a loop, a terminating loop and a loop formula for an $smodels$ program, with the completion and loop formulas being formulas in the logic PL^{wa}.

Results in [4] imply that stable models of an *smodels* program are in one-to-one correspondence with models (in the sense of the logic PL^{wa}) of the completion of the program extended with some loop formulas for the program. That result, generalizing a result of Lin and Zhao [2], allows us to extend the design of $assat$ to the case of *smodels* programs in a way, which does not require that weight atoms be complied away. In a nutshell, we first compute the completion of the $smodels$ program. Then we iteratively compute models of the completion using PB solvers. Whenever a non-stable model is found, we add to the completion loop formulas that guarantee that the same non-stable model will not be computed again. Our algorithm is shown in Figure 1.

Input: P — a ground logic program (possibly with weight atoms)
A — a pseudo-boolean solver
Output: M — a stable model of P if A finds one; "Failed" otherwise

BEGIN
compute the completion $comp(P)$ of P represented in PB logic;
do
M := a model of $comp(P)$ found by A;
if (M does not exist) output "Failed" and terminate;
if (M is stable) output M and terminate;
compute the reduct P^M of P with respect to M;
compute the greatest stable model M' under M in P^M;
M^- := $M \setminus M'$;
find all maximal loops in M^-;
add loop formulas of the loops found in previous step to $comp(P)$;
while (**true**);
END

Fig. 1. Algorithm of $pbmodels$

3 *Pbmodels* — the Package

The *pbmodels* package, including executable PB solvers code, can be obtained at `http://www.cs.uky.edu/ai/pbmodels/pbmodels-0.1.tar.gz`. It is also installed at the *asparagus* site `http://asparagus.cs.uni-potsdam.de`.

The package contains the source code of `pbmodels` and supported PB solvers: *satzoo* [5], *pbs* [6], *wsatoip* [7], and *wsatcc* [8]. The first two are complete PB solvers while the last two are incomplete PB solvers based on *walksat* [9] local-search algorithm. In addition, the package contains also two scripts:

- `esrapl`: a perl script that recovers the structure of rules and weight atoms when `lparse` grounds the input program and converts it to a normal form required by *smodels*. We found that PB solvers are more effective when the original structure of rules and weight atoms is restored. We designed `esrapl`[1] to "undo" conversion of ground programs to their normal form by `lparse`.

[1] `lparse` spelled backwards.

– `convert2`: a perl script that takes a theory in the logic PL^{wa}, and produces PB theories that are accepted by different PB solvers. It is invoked by the command:
`convert2 <target format> <source file>`
Values for `<target format>` are: `satzoo`, `pbs`, `wsatcc`, or `wsatoip`. They specify the format, to which the input PL^{wa} theory in the file `<source file>` is to be converted.
The script can be used as a stand alone format translation program.

4 Input, Output, and How to Invoke *Pbmodels*

Pbmodels accepts on input programs obtained by grounding *smodels* programs with `lparse` and processing the result with `esrapl`. The output of *pbmodels* is similar to that of *smodels*. The string "False" indicates that there are no stable models for the input program[2]. Otherwise, the first stable model of the input will be printed on the screen.

The main options for `pbmodels` are:

1. `--engine <engine name>`
   It specifies which PB solver to use. Values for this option are: `satzoo`, `pbs`, `wsatcc`, and `wsatoip`
2. `--option <list of options to the solver>`
   It specifies options for a solver selected with the `--engine` option. Everything after `--option` is passed to the solver.

In addition, there are also options that allow the user to specify the path to executable programs if they are not installed to the default bin directory. We do not discuss them here due to space constraints.

The following examples show how to invoke *pbmodels*. We assume that the file *prog.lp* contains an input *smodels* program. We also assume that *satzoo* is to be used as the back-end engine.

```
lparse prog.lp | esrapl | pbmodels --engine satzoo
```

The following command will invoke *pbmodels*, use *satzoo*, and pass specific *satzoo* options to *satzoo*:

```
lparse prog.lp | esrapl | pbmodels --engine satzoo --option
-no-rand
```

5 Performance

We report here briefly on our experiments comparing the performance of *pbmodels* and *smodels*. We considered several benchmark problems. Due to lack of space, we present here only the results concerning the *traveling salesperson problem*, and the *weighted n-queens problem*. Specifications of both problems use general weight atoms. Additional experimental results can be found at `http://www.cs.uky.edu/ai/pbmodels`.

We tested *pbmodels* with the four PB solvers mentioned earlier and compared the results with those obtained for *smodels*. All experiments were run on machines with

[2] If an *incomplete* solver is used, "False" means that no stable models were found; the program may actually have stable models.

3.2GHz Pentium 4 CPU, 1GB memory, running Linux with kernel version 2.6.11, gcc version 3.3.4.

In the following tables we report the number of instances a solver solved, and the number of times a solver won among all solvers, and among the complete solvers (that latter metric for complete solvers only). We also report the average and the median running times in seconds, over all instances that do not time out[3].

Table 1 shows results on the traveling salesperson problem. We randomly generated 50 weighted complete graphs containing 20 vertices. For each of them, we set the upper bound on the length of the TSP cycle to $w = 62$. The value of w is chosen so that about half of the instances have solutions [4].

TSP $(n = 20, w = 62)$	*# of Instances* *Solved*	*# of Times Won* *v.s. All*	*v.s. Complete*	*Timing* *Mean*	*Median*
$smodels$	$19/50$	1	7	1558.37	1637.14
$pbmodels\text{-}satzoo$	$19/50$	2	16	696.42	461.25
$pbmodels\text{-}pbs$	$1/50$	0	0	1482.24	1482.24
$pbmodels\text{-}wsatcc$	$19/50$	6	–	28.39	6.59
$pbmodels\text{-}wsatoip$	$28/50$	22	–	7.20	1.43

Table 1. TSP Problem

Among complete solvers, $pbmodels\text{-}satzoo$ performs better than the other two. Even though $pbmodels\text{-}satzoo$ and $smodels$ solved the same number of instances, $pbmodels\text{-}satzoo$ won more times among the three complete solvers. Moreover, the average running time of $pbmodels\text{-}satzoo$ is about half of that of $smodels$ and the median running time is about $1/3$ of that of $smodels$. Over all, $pbmodels\text{-}wsatoip$ is the winner. It solves the largest number of instances. Furthermore, its average and median running times are about three orders of magnitude less than the running time of $smodels$.

Table 2 shows the results on the *weighted n-queens problem*. An instance to this problem consists of n^2 non-negative integer weights, one for each square of an $n \times n$ chessboard, and of an integer bound w. A solution to an instance is a placement of n queens on the chessboard so that they do not attack each other and the total weight of the placement (the sum of the weights of the squares occupied by queens) is no greater than w. We randomly generated 50 weighted "chessboards" of the size 20×20 and, in each case, we set $w = 50$ as for $w = 50$ about half of the instances have solutions.

We observe that $pbmodels\text{-}wsatcc$ and $pbmodels\text{-}wsatoip$ outperformed $smodels$. Among the complete solvers, $smodels$ is slightly better than $pbmodels\text{-}satzoo$ and *pbmodels-pbs* ($smodels$ managed to solve two instances in this category, other complete solvers timed-out on all). However, the local-search solvers are the overall winners in all metrics considered.

[3] We do not include the time used by `lparse` and `esrapl` in the time reported, as we are only interested in the effectiveness of solvers.

[4] To be precise, we do not know that the remaining ones do not have solutions. We only know that no solver was able to solve them within the 3000 seconds time limit we set.

W-NQueens ($n = 20, w = 50$)	*# of Instances Solved*	*# of Times Won*		*Timing*	
		v.s. All	*v.s. Complete*	*Mean*	*Median*
smodels	$2/50$	0	2	697.81	661.20
pbmodels-satzoo	$0/50$	0	0	N/A	N/A
pbmodels-pbs	$0/50$	0	0	N/A	N/A
pbmodels-wsatcc	$29/50$	15	–	1.01	0.33
pbmodels-wsatoip	$29/50$	14	–	0.44	0.35

Table 2. Weighted NQueens Problem

6 Conclusions

We have presented software package *pbmodels* that computes models of *smodels* programs. The key feature of our system is that it supports the use of off-the-shelf *PB* solvers developed by the satisfiability community and capable of process weight atoms directly.

Our experiments show that *pbmodels* performs better than *smodels* on benchmarks we considered. The results were especially good when local-search *PB* solvers were used. We observed the same phenomenon in experiments on other problems we considered: weighted Latin square problem, magic square problem, vertex-cover problem and the tower-of-Hanoi problem. The last two problems use only cardinality atoms. Hence we were able to include *cmodels* [3] in those two tests, too. The results showed that *PB* solvers are generally faster than *cmodels* on these two benchmark families.

Acknowledgments

We acknowledge the support of NSF grants IIS-0097278 and IIS-0325063.

References

1. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234.
2. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. In: Proceedings of AAAI-2002, AAAI Press (2002) 112–117.
3. Babovich, Y., Lifschitz, V.: *Cmodels package* (2002) `http://www.cs.utexas.edu/users/tag/cmodels.html`.
4. Liu, L., Truszczyński, M.: Properties of programs with monotone and convex constraints. In: Proceedings of AAAI-05, AAAI Press (2005).
5. Eén, N., Sörensson, N.: An extensible SAT solver. In: Proceedings of SAT-2003. Volume 2919 of LNCS., Springer (2003) 502–518.
6. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: PBS: a backtrack-search pseudo-boolean solver and optimizer. In: Proceedings of SAT-2003. 346 – 353.
7. Walser, J.: Solving linear pseudo-boolean constraints with local search. In: Proceedings of AAAI-97, AAAI Press (1997) 269–274.
8. Liu, L., Truszczyński, M.: Local-search techniques in propositional logic extended with cardinality atoms. In: Proceedings of CP-2003. Volume 2833 of LNCS, Springer (2003) 495–509.
9. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of AAAI-1994, AAAI Press (1994) 337–343.