

***WSAT(CC)* — a fast local-search ASP solver**

Lengning Liu and Mirosław Truszczyński

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

Abstract. We describe *WSAT(CC)*, a local-search solver for computing models of theories in the language of propositional logic extended by cardinality atoms. *WSAT(CC)* is a processing back-end for the logic *PS⁺*, a recently proposed formalism for answer-set programming.

1 Introduction

WSAT(CC) is a local-search solver for computing models of theories in the logic *PL^{cc}*, the propositional logic extended by cardinality atoms [3,4]. It can serve as a processing back-end for the logic *PS⁺* [3], an *answer-set programming* (ASP) formalism based on the language of predicate calculus and, hence, different from typical ASP systems that have origins in logic programming.

A *clause* in the logic *PL^{cc}* is a formula $\alpha_1 \wedge \dots \wedge \alpha_r \rightarrow \alpha_{r+1} \vee \dots \vee \alpha_s$, where each α_i , $1 \leq i \leq s$, is a propositional atom, or a *cardinality atom* (*c-atom*, for short) — an expression $k\{a_1, \dots, a_n\}m$, where a_i are propositional atoms and k, m and n are integers such that $0 \leq k \leq m \leq n$. A set of atoms M is a *model* of a c-atom $k\{a_1, \dots, a_n\}m$ if $k \leq |M \cap \{a_1, \dots, a_n\}| \leq m$. With this definition, the semantics of clauses and theories in the logic *PL^{cc}* is a straightforward extension of the semantics of propositional logic. *PL^{cc}* theories arise by grounding theories in the ASP logic *PS⁺* by means of the grounder program *asppsgrnd* [3].

We discuss here an implementation of *WSAT(CC)*. We restrict the discussion to most essential concepts and options only. For more details and bibliography on related work on propositional logic extended by c-atoms and pseudo-boolean constraints, we refer to [4], which introduced *WSAT(CC)*, and to [2].

2 *WSAT(CC)* — a brief description and a list of options

As other *WSAT*-like local-search solvers [6,7], *WSAT(CC)* searches for models in a series of *tries*, starting with a random assignment of truth values to atoms. Each try consists of steps, called *flips*, which produce "new" truth assignments by *flipping* the truth values of some of the atoms. If a flip produces a satisfying assignment, this try is terminated and another one starts. *WSAT(CC)* supports several strategies to select atoms for flipping. All of them require a parameter called the *noise level*. It determines the probability of applying a random walk step in order to escape from a local minimum. The maximum numbers of tries and flips, and the noise level are set from the command line by means of the options **-t**, **-c** and **-N**, respectively.

WSAT(CC) is different from other similar algorithms in the way in which it computes the *break-count* of an atom (used to decide which atom to flip) and in the way it executes a flip. The choice of the break-count computation method or of the way a flip is defined determines a particular local-search strategy in *WSAT(CC)*. At present, *WSAT(CC)* supports three basic methods.

Virtual break-count. We define virtual break-counts with respect to a propositional theory, in which c-atoms are replaced by their equivalent propositional representations. However, in the actual computation we use the original theory (with c-atoms) rather than its propositional-logic counterpart (with c-atoms removed), as the latter is usually exponentially larger. To invoke virtual break-count method, we use the option `-VB`. The virtual break-count method is applicable with all PL^{cc} theories and is a default method of *WSAT(CC)*.

Double flip. It applies only to *simple* PL^{cc} theories that are specified by the following two conditions: (a) all the c-atoms appear in unit clauses, and (b) all the sets of atoms in the c-atoms are pairwise disjoint. A flip is designed so that all unit clauses built of c-atoms remain satisfied. Thus, on occasion, two atoms will change their truth values in one flip step. The break-count is defined with respect to regular propositional clauses as in *WSAT*. To invoke this method, we use the option `-DF`.

Permutation flip. It applies to theories, in which c-atoms are used solely to specify permutations (for instance, when defining an assignment of queens in the n -queens problem). Flips realize an inverse operation on permutations and, hence, transform a permutation into another permutation. As a consequence, all unit clauses built of c-atoms are always satisfied. To accomplish that, four atoms must have their truth values changed in one flip step. The break-count is defined with respect to regular propositional clauses of the theory in the same way as in *WSAT*. We invoke this method with the option `-PF`.

3 *WSAT(CC)* — input, output and how to invoke it

WSAT(CC) accepts input files containing PL^{cc} theories described in a format patterned after that of CNF DIMACS. The first line is of the form `p <na> <nc>`, where `na` and `nc` are the number of propositional atoms and clauses in the theory, respectively. The following lines list clauses. A clause $\alpha_1 \wedge \dots \wedge \alpha_r \rightarrow \alpha_{r+1} \vee \dots \vee \alpha_s$, is written as `A1 ... Ar , A(r+1) ... As`, where each `Ai` is a positive integer (representing the corresponding atom α_i), or an expression of the form `{k m C1 ... Cn}` (representing a c-atom $k\{a_1, \dots, a_n\}m$).

WSAT(CC) outputs models that it finds as well as several statistics to standard output device (or, depending on the options used, to a file in a user-readable format). It also creates a file `wsatcc.stat` that stores records summarizing every call to *WSAT(CC)* and key statistics pertaining to the computation.

Typical call to *WSAT(CC)* looks as follows: `wsatcc -f file -t 200 -c 150000 -N 10 100`. It results in *WSAT(CC)* looking for models to the PL^{cc} theory specified in `file`, by running 200 tries, each consisting of 150000 flips. The noise level is set at 10/100 (=0.1).

4 *WSAT(CC)* package

WSAT(CC) solver and several related utilities can be obtained from <http://www.cs.uky.edu/ai/wsattcc/>. *WSAT(CC)* works on most Unix-like operating systems that provide gcc compiler. The utilities require Perl 5 or greater. For more details on installation, we refer to [2].

5 Performance

Our experiments demonstrate that *WSAT(CC)* is an effective tool to compute models of *satisfiable* PL^{cc} theories and can be used as a processing back-end for the ASP logic PS^+ . In [4], we showed that *WSAT(CC)* is often much faster than a local-search SAT solver *WSAT* and has, in general, a higher success rate (likelihood that it will find a model if an input theory has one). In [1], we used *WSAT(CC)* to compute several new lower bounds for van der Waerden numbers. Here, we will discuss our recent comparisons of *WSAT(CC)* with *WSAT(OIP)* [7], a solver for propositional theories extended with pseudo-boolean constraints (for which we developed utilities allowing it to accept PL^{cc} theories).

We tested these programs on PL^{cc} theories encoding instances of the vertex-cover and *open n-queens* problems¹. We generated these theories by grounding appropriate PS^+ theories extended with randomly generated problem instances.

Table 1 shows results obtained by running *WSAT(CC)* (both *-VB* and *-DF* versions are applicable in this case) and *WSAT(OIP)* to find vertex covers of sizes 1035, 1040 and 1045 in graphs with 2000 vertices and 4000 edges. The first column shows the size of the desired vertex cover and the number of graphs (out of 50 that we generated), for which we were able to find a solution by means of at least one of the methods used. The remaining columns summarize the performance of the three algorithms used: *WSAT(CC)-VB*, *WSAT(CC)-DF*, and *WSAT(OIP)*. The entries show the *time*, in seconds, needed to complete computation for all 50 instances and the *success rate* (the percentage of cases where the method finds a solution to all the instances, for which at least one method found a solution).

Table 1. Vertex cover: Large Graphs

Family	<i>WSAT(CC)-VB</i>	<i>WSAT(CC)-DF</i>	<i>WSAT(OIP)</i>
1035 (9 / 50)	1453/77%	3426/100%	9748/11%
1040 (24 / 50)	1166/95%	2464/100%	7551/100%
1045 (36 / 50)	991/86%	1610/100%	6365/100%

The results show that *WSAT(CC)-VB* is faster than *WSAT(CC)-DF*, which in turn is faster than *WSAT(OIP)*. However, *WSAT(CC)-VB* has generally the lowest success rate while *WSAT(CC)-DF*, the highest.

We note that we attempted to compare *WSAT(CC)* with *smodels* [5], a leading ASP system. We found that for the large instances that we experimented

¹ In the open *n*-queens problem, given an initial “attack-free” assignment of *k* ($k < n$) queens on the $n \times n$ board, the goal is to assign the remaining $n - k$ queens so that the resulting assignment is also “attack-free”.

with *smodels* failed to terminate within the time limit that we allocated per instance. That is not surprising, as the search space is prohibitively large for a complete method and *smodels* is a complete solver.

The open n -queens problem allowed us to experiment with the method *-PF* (permutation flip). It proved extremely effective. We tested it for the case of 50 queens with 10 of them preassigned. We generated 100 random preassignments of 10 queens to a 50×50 board and found that 55 of them are satisfiable. We tested the four algorithms only on those satisfiable instances. The results are shown in Table 2.

Table 2. Open n -Queens: $N = 50$, 10 preassigned

Family	<i>WSAT(CC)-VB</i>	<i>WSAT(CC)-PF</i>	<i>WSAT(OIP)</i>	<i>smodels</i>
50+10(55 / 55)	20/1539/100%	9/768/100%	76/1459/100%	908/10%

Here, we include another measurement for local search solvers. The second number shows the average number of flips each method uses in finding one solution. *WSAT(CC)-VB* is faster than *WSAT(OIP)* even though they have the similar number of flips. *WSAT(CC)-PF* is even more powerful because it uses the fewest number of flip and is the fastest. *Smodels* can only find solutions for 6 instances within the 1000-second limit and turns out to be the slowest.

We tested the version *-PF* with one of the encodings of the Hamiltonian-cycle problem and discovered it is much less effective there. Conditions under which the version *-PF* is effective remain to be studied.

Acknowledgments

This research was supported by the National Science Foundation under Grants No. 0097278 and 0325063.

References

1. M.R. Dransfield, V.M. Marek, and M. Truszczyński. Satisfiability and computing van der Waerden numbers. In *Proceedings of SAT-2003*. LNAI, Springer Verlag, 2003.
2. D. East, L. Liu, S. Logsdon, V. Marek, and M. Truszczyński. ASPPS user’s manual, 2003. http://www.cs.uky.edu/aspps/users_manual.ps.
3. D. East and M. Truszczyński. Propositional satisfiability in answer-set programming. In *Proceedings of KI-2001*, LNAI 2174. Springer Verlag, 2001. Full version submitted for publication (available at <http://xxx.lanl.gov/abs/cs.L0/0211033>).
4. L. Liu and M. Truszczyński. Local-search techniques in propositional logic extended with cardinality atoms. In *Proceedings of CP-2003*. LNCS, Springer Verlag, 2003.
5. I. Niemelä and P. Simons. Extending the *smodels* system with cardinality and weight constraints. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer Academic Publishers, 2000.
6. B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of AAAI-94*. AAAI Press, 1994.
7. J.P. Walser. Solving linear pseudo-boolean constraints with local search. In *Proceedings of AAAI-97*. AAAI Press, 1997.