2005

Cmodels for Tight Disjunctive Logic Programs

Yuliya Lierler
University of Nebraska at Omaha, ylierler@unomaha.edu

Cmodels for Tight Disjunctive Logic Programs*

Yuliya Lierler

AI, Erlangen-Nürnberg Universität,
`yuliya.lierler@informatik.uni-erlangen.de`

1 Introduction

Disjunctive logic programming under the stable model semantics [GL91] is a new *answer set programming* (ASP) methodology for solving combinatorial search problems. It is a form of declarative programming related to logic programming languages, such as Prolog, where the solutions to a problem are represented by answer sets, and not by answer substitutions produced in response to a query as in convential logic programming. Instead of Prolog systems, this programming method uses answer set solvers, such as SMODELS[1], SMODELScc[2], CMODELS[3], DLV[4], and GNT[1]. These systems made it possible for ASP to be successfully applied in such areas as planning, bounded model checking, and space shuttle control. DLV and GNT are more general as they work with the class of disjunctive logic programs, while other systems cover nondisjunctive programs. System CMODELS uses SAT solvers as search engines, which allows it to take advantage of rapid progress in the area of SAT. CMODELS proved to be an efficient system in providing the solution to the wire-routing problem [EW04], and to the problem of reconstructing probable phylogenies in the area of historical linguistics [BEMR05]. In this work we extend CMODELS [GLM04] to tight disjunctive programs. Complexity of finding a solution for such programs is NP, as in the case of nondisjunctive programs. Extending the syntax of the input language of CMODELS to tight disjunctive programs permits the knowledge engineer to be more flexible with the encoding of the problems in the NP complexity class. Experimental analyses demonstrate that the approach is computationally promising and may advance applications of disjunctive logic programming.

2 Theory, Implementation, Usage, Experiments

We base our work on the relationship between the completion [Cla78] and answer set semantics for logic programs. For the large class of *tight* programs the answer

* I would like to thank V. Lifschitz for many valuable suggestions for the format of this paper and E. Giunchiglia, G. Görz, J. Lee, and M. Maratea for the comments related to the subject.

[1] `http://www.tcs.hut.fi/Software/` .

[2] `http://www.ececs.uc.edu/~ schlipf/` .

[3] `http://www.cs.utexas.edu/users/tag/cmodels` .

[4] `http://www.dbai.tuwien.ac.at/proj/dlv/` .

```
% Sample graph encoding, i.e. graph contains 3 nodes, and 3 edges:
% edges between nodes 1 and 2, 2 and 3, 3 and 1.
node(1..3). edge(1,2).edge(2,3).edge(3,1).
% Declaration of three colors
col(red). col(green). col(blue).
% Disjunctive rule: stating that node has some color
colored(X,red) | colored(X,green) | colored(X,blue) :- node(X).
% Neighboring nodes should not have the same color
:- edge(X,Y), colored(X,C), colored(Y,C), col(C).
```

Fig. 1. Encoding of tight 3-colorability problem for grounder LPARSE: *3-col.lp*

sets of the program are the same as the models of its completion, and hence SAT solvers can play the role of answer set enumerators. [LL03] introduced the notion of completion and tightness for disjunctive programs. A *disjunctive program* Π is a set of disjunctive rules of the form $A \leftarrow B, F$ where A is the head of the rule, and is either a disjunction of atoms or symbol $\perp$, B is a conjunction of atoms, and F is a formula of the form $not\ a_1, \ldots, not\ a_m$. We identify the disjunction of atoms A with the set of the atoms occurring in A. The completion of Π [LL03] is the set of propositional formulas that consists of the implication $B \wedge F \supset A$ for every rule in Π, and the implication $a \supset \bigvee_{A \leftarrow B,F \in \Pi;\ a \in A} (B \wedge F \wedge \bigwedge_{p \in A \setminus \{a\}} \neg p)$ for each atom $a \in \Pi$. The *positive dependency graph* of Π is directed graph G such that the vertices of G are the atoms occurring in Π, and for every rule in Π, G has an edge from each atom in A to each atom in B. Program Π is *tight* if its positive dependency graph is acyclic.

Theorem. [LL03] *For any tight disjunctive program Π and any set X of atoms, X is an answer set for Π iff X satisfies the completion of Π.*

Figure 1 presents the tight disjunctive program *3-col.lp* based on the encoding of 3-colorabilty problem provided at the DLV web page.

We based our implementation on systems LPARSE *--dlp* and CMODELS. LPARSE *--dlp* takes a disjunctive logic program with variables as an input and grounds the program. In order to use CMODELS for solving disjunctive programs flag *-dlp* should be used. In the process of its operation, CMODELS *-dlp* first verifies that the program is tight, by building the positive dependency graph and using a depth first search algorithm to detect a cycle in it. This step may be omitted from the execution sequence using flag *-t*. Second, CMODELS *-dlp* forms the program's completion, and last it calls a SAT solver to find the models of the completion. Flags *number, -si, -rs, -mc* are available, where *number* is an integer that stands for a number of solutions to find (0 stands for all solutions, 1 is a default), and *-mc (default), -si, -rs* specify that SAT solver CHAFF, SIMO, RELSAT, respectively, is invoked during the search. For example, command line

```
lparse --dlp 3-col.lp | cmodels -dlp
```

produces one answer set for the program in Figure 1:

```
Answer set: colored(1,red) colored(2,green) colored(3,blue)
```

It is worth noticing that *3-col.lp* program is syntactically identical to the 3-colorability program with choice rules supported by systems SMODELS, SMODELScc and CMODELS. The disjunctive rule of program *3-col.lp* is interpreted as the choice rule by these systems. Semantically, the rules are nevertheless different. The choice rule encodes the exclusive disjunction in the head of this rule. In case of 3-colorabilty problem this is acceptable interpretation of a rule and this allows us to find answer sets of the program also by means of nondisjunctive answer set programming.

For experimental analyses we used the encoding of the 3-colorability problem as in Figure 1. We compared the performance of CMODELS *–dlp* with systems DLV, GNT on disjunctive program and also SMODELS, SMODELScc and CMODELS on choice rule encoding of a problem. All experiments were run on Pentium 4, CPU 3.00GHz and presented in Figures 2 and 3.

In Figure 2 we show the results of running CMODELS with SIMO and DLV on the disjunctive programs, and SMODELS on the corresponding program with choice rules. The instances of Ladder graphs presented in the table were taken from the DLV web page. Columns LPARSE *–dlp*, CMODELS, DLV and SMODELS present the running times of the systems in seconds. DLV running time also includes the time spent by the system on grounding the program. We can see that CMODELS outperforms two other systems by more than an order of magnitude.

In Figure 3 we present the experiments with harder instances of the graphs. Letters L, S in the names of the graph instances stand for ladder and simplex, while the number stands for the number of nodes divided by 1000 in the ladder graphs, and the number of levels in the simplex graphs. SMODELS, DLV and GNT were not able to terminate on our test programs within the 30 minutes cutoff time. Columns LPARSE, CMODELS *simo*,CMODELS *chaff*, and SMODELScc present the running times of the systems in seconds. The numbers before and after "\" stand for invocation of LPARSE *--dlp*, CMODELS *-dlp* on disjunctive programs, and LPARSE, CMODELS on corresponding programs with choice rules, respectively. The second and the third columns demonstrate that the ground disjunctive program is smaller than the corresponding ground program with choice rules: LPARSE encodes disjunctive rules more economically than choice rules. CMODELS *-dlp*, in its turn, takes an advantage of a smaller ground program and produces fewer clauses. For example, the performance of CMODELS *-dlp simo* on the disjunctive program saves 18 to 29% of running time in comparison with CMODELS *simo* performance on the program with choice rules. CMODELS *-dlp simo* also outperforms SMODELScc on ladder graphs by an order of magnitude. In case of simplex graph instances CMODELS *-dlp chaff* also outperforms SMODELScc. Capability of using different search engines may prove to be useful in practical applications.

3 Conclusions and Future Work

The evaluation of CMODELS *-dlp* shows that it is a promising approach that might advance the use of the disjunctive answer set programming paradigm in

# of nodes	LPARSE *-dlp*	CMODELS *simo*	SMODELS	DLV
1080	0.06	0.10	2.42	4.00
1320	0.07	0.12	3.74	6.00
1680	0.10	0.17	5.94	9.53
1920	0.12	0.19	7.91	12.46
2400	0.14	0.25	11.97	19.98

Fig. 2. 3-colorability problem on Ladder graphs: CMODELS *-dlp* with SAT solver SIMO on disjunctive program vs. SMODELS on choice rule program and DLV

Pr.	LPARSE	# rules $*10^{-4}$	# clauses $*10^{-4}$	CMODELS *simo*	CMODELS *chaff*	SMODELScc
	disj\ch	disj\ch	disj\ch	disj\ch	disjunctive	choice
L32	2\3	25\38	44\51	10\15	27	105
L64	4\6	51\76	89\102	36\51	157	417
L120	8\11	96\144	168\192	122\165	727	1460
L240	16\24	192\288	336\384	496\701	-	-
S480	14\17	161\207	230\253	174\213	20	26
S600	21\27	251\323	359\395	378\490	35	46

Fig. 3. 3-colorability problem on large Ladder and Simplex graphs: CMODELS *-dlp* with SAT solvers SIMO and CHAFF on disjunctive programs vs. CMODELS with SIMO and SMODELScc on choice rule programs

practice. [LZ02] provided the theoretical base for using SAT solvers for computing answer sets for nontight nondisjunctive programs. Systems ASSAT [LZ02] and CMODELS [GLM04] are the implementations that demonstrated promising experimental results. [LL03] extended the theory used by the approach to the case of nontight disjunctive programs. Future work is to add the capability to CMODELS to find answer sets for nontight disjunctive programs.

References

[BEMR05] D. R. Brooks, E. Erdem, J. W. Minett, and D. Ringe. Character-based cladistics and answer set programming. In *Proc. PADL'05*, pages 37–51, 2005.

[Cla78] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[EW04] E. Erdem and M.D.F. Wong. Rectilinear steiner tree construction using answer set programming. In *Proc. ICLP'04*, pages 386–399, 2004.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GLM04] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. Sat-based answer set programming. In *Proc. AAAI-04*, pages 61–66, 2004.

[LL03] Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *Proc. ICLP-03*, pages 451–465, 2003.

[LZ02] Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proc. AAAI-02*, pages 112–117, 2002.