

**Confronto tra gli approcci CLP(FD) e ASP ai problemi NP
completi**
*A comparison of CLP(FD) and ASP solutions to NP-complete
problems*

Agostino Dovier

Andrea Formisano

Enrico Pontelli

SOMMARIO/ABSTRACT

Programmi CLP(FD) e programmi logici normali nel contesto dell'Answer Set Programming sono probabilmente i due principali approcci dichiarativi per modellare e risolvere problemi NP-completi. In questo lavoro vengono comparati i risultati computazionali e la naturalezza dei due paradigmi per risolvere alcuni problemi NP-completi. *CLP(FD) and normal logic programs in the context of Answer Set Programming are perhaps the two main declarative approaches to modeling and solving NP-complete problems. In this paper, we compare the encoding style and computational results for the two paradigms, studying some NP-complete problems.*

Keywords: CLP(FD), ASP, NP-completeness

1 Introduction

It is well-known [4] that, given a propositional normal logic program P , deciding whether or not it admits a *stable model* (a.k.a. an *answer set*) [2] is a NP-complete problem. As a consequence, any NP-complete problem can be encoded as a normal logic program under answer set semantics (also known as *Answer Set Programming (ASP)*). Answer-set solvers (e.g., Smodels, Cmodels, ASSAT, DLV [5]) are programs designed for computing the answer sets of normal logic programs. These tools can be seen as theorem provers, or model builders, enhanced with several built-in heuristics, aimed at pruning the solution's space. Most of them rely on variations of the Davis-Putnam-Longeman-Loveland procedure in their computations. Such solvers are often equipped with a front-end that transforms a collection of non propositional normal clauses (without function symbols, or with limited uses of them) in a *finite* set of ground instances of such clauses. Use of function symbols is limited to situations where finite grounding is guaranteed. Some solvers (e.g., Smodels) provide some forms of *optimization statements*: the solver

will look for an answer set that maximizes (or minimizes) an objective function dependent on the content of the answer set.

An alternative framework that is frequently adopted to handle NP-complete problems is the *Constraint Logic Programming over Finite Domains (CLP(FD))* framework [3]. In this context, a finite domain of objects (typically integers) is associated to each variable in the problem, and the typical constraints are literals of the forms $s = t$, $s \neq t$, $s < t$, $s \leq t$, where s and t are arithmetic expressions. In spite of the expressive power of this framework, which has been shown to be Turing-complete, both the encoding of search problems and the control of the search-tree exploration come very naturally by using constraints. Indeed, a large literature has been developed presenting applications of CLP(FD) to a variety of search and optimization problems.

In this paper we report on a preliminary comparison of these two declarative approaches on a number of NP-complete problems.

2 The experimental framework

We formalized the problems both in CLP(FD) and in ASP.

The CLP code was intended to be executed either by SICStus Prolog (using the library `clpfd`) or by ECLiPSe (using the library `ic`) [6]. All codes and results can be found in <http://www.di.univaq.it/~formisano/CLPASP>. The ASP code was tailored to be processed by `lparse+smodels` and `lparse+cmodels` [5]—work is in progress to extend the experiments to other ASP solvers. Taking fully advantage of the logic framework, we encoded the various problems in the more declarative way possible.

We focused on well-known computationally-hard problems. Among them:

- Graph k -coloring
- Hamiltonian circuit

- Schur numbers (the Schur number $S(P)$ is the largest integer N for which the interval $[1..N]$ can be partitioned into P sum-free sets. A sum-free set S is a set for which the intersection of S and $\{x + y : x \in S, y \in S\}$ is empty.)
- Protein structure prediction on a 2D lattice (it is a simplified version of the protein folding problem [1]).
- Planning in a block world (given N blocks $1, \dots, N$, in the *initial state*, the blocks are arranged in a single stack, in increasing order (block 1 is on the table, Block N is on top of the stack). In the *goal state*, there must be two stacks, composed of the blocks with odd and even numbers, respectively, both of them increasing order. The planning problem consists of finding a sequence of T actions to reach the goal state, starting from the initial state. Some additional restrictions must be met: first, in each state at most three blocks can lie on the table; moreover, a block x cannot be placed on top of a block y if $y \geq x$.)
- Generalized Knapsack (given n types of objects, where each object of type i has size w_i and it costs c_i , establish whether there is a way to fill a knapsack with X_1 object of type 1, X_2 objects of type 2, and so on, so that: $\sum_{i=1}^n X_i w_i \leq \text{max_size}$ and $\sum_{i=1}^n X_i c_i \geq \text{min_profit}$, where max_size is the capacity of the knapsack and min_profit is the minimum profit required.)

Some of the programs have been drawn from the best proposals appeared in the literature, others are novel solutions, developed in this project (e.g., the ASP implementation of the Protein structure prediction problem and the planning implementation in CLP(FD)). The CLP programs have been written faithfully following the traditional constraint-&-generate style.

As concerns graphs problems, our major source of case studies was the repository of ‘‘Graph Coloring and its Generalizations’’ [7], which provides the community a varied collection of instances mainly aimed at benchmarking algorithms and approaches to graph problems.

3 Results and Conclusions

As far as the experimental results are concerned, we report only the results obtained using SICStus Prolog—in the context of our problems, the ECLiPSe CLP(FD) engine performed on average 10 times slower than SICStus Prolog. The timings have been obtained measuring only the actual computation of the solution (e.g., ignoring the time to read the inputs and the time to ground the program—in the case of ASP).

First of all, we wish to notice that ASP provides a more compact, and probably more declarative, encoding. As far as running times are concerned, CLP(FD) definitely wins the comparison vs. ASP-SModels. The comparison between CLP(FD) and ASP-CModels is more interesting.

	Col.	Ham.	Schur	PF	Plan.	Knap.
CLP(FD)	+	++	+	+	+	+
ASP	++	+	++	-	+	-

Table 1: Schematic results’ analysis.

Table 1 intuitively summarizes our observations drawn from the different benchmarks. Although these experiments are quite preliminary, they actually provide already some concrete indications that can be taken into account when choosing a paradigm to tackle a problem. We can summarize the main points as follows:

- graph-based problems have nice compact encodings in ASP and the performance of the ASP solutions is acceptable and scalable;
- problems requiring more intense use of arithmetic and/or numbers are declaratively and efficiently handled by CLP(FD);
- for problems with few arithmetic, the exponential growth w.r.t. input size is less evident for ASP.

In the future we plan to extend our analysis to other problems and to other constraint solvers. We are interested in assessing if it is possible to formalize domain and problem characteristics to lead the choice of which paradigm to use, and, if it is possible to introduce strategies to identify problem components and map them to cooperating solvers (using the best solver for each part of the problem). In particular, we are interested in characterizing those contexts where the ASP solvers perform significantly better than CLP (e.g., problems with incomplete information).

Acknowledgements. This work is partially supported by GNCS2005 project on constraints and their applications.

REFERENCES

- [1] P. Clote and R. Backofen. *Computational molecular biology: An introduction*. Wiley, 2001.
- [2] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of the 5th ICLP*, pp.1070–1080, 1988. The MIT Press.
- [3] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *J. of Logic Programming*, 19/20:503–581, 1994.
- [4] W. Marek and M. Truszczyński. Autoepistemic logic. *J. of the ACM*, 38(3):588–619, 1991.
- [5] Web references for ASP solvers. ASSAT: assat.cs.ust.hk. CCalc: www.cs.utexas.edu/users/tag/cc. CModels: www.cs.utexas.edu/users/tag/cmodels. DeReS and aspps: www.cs.uky.edu/ai. DLV: www.dbai.tuwien.ac.at/proj/dlv. SModels: www.tcs.hut.fi/Software/smodels.
- [6] Web references for some CLP implementations: ECLiPSe: www.icparc.ic.ac.uk/eclipse SICStus: www.sics.se/sicstus.
- [7] Web site of COLOR02/03/04: Graph Coloring and its Applications: mat.gsia.cmu.edu/COLORING03.

4 Contacts

Agostino Dovier
Università di Udine, Dipartimento di Matematica e Informatica.
Via delle Scienze 206, 33100 Udine (Italy).
Tel: +39 0432 558494
E-mail: dovier@dimi.uniud.it.

Andrea Formisano (Corresponding Author)
Università di L'Aquila, Dipartimento di Informatica.
Via Vetoio-loc. Coppito, 67010 L'Aquila (Italy)
Tel: +39 0862 433740
E-mail: formisano@di.univaq.it

Enrico Pontelli
New Mexico State University, Department of Computer Science.
Las Cruces, NM 88003-0001, USA
Tel. +1 505-646-6239
E-mail: epontell@cs.nmsu.edu

5 Biography

Agostino Dovier received his Laurea degree in Computer Science from the University of Udine and the PhD degree in Computer Science from the University of Pisa. He is an associate professor in Computer Science at the University of Udine. His research interests include constraint programming language design, programming with sets and multisets, computable set theory, Web-like databases and related graph algorithms, and computational biology. He has been part of program committees of national and international conferences on declarative programming and he is coordinator of two research projects on constraint programming and its applications. He is a member of AI*IA.

Andrea Formisano received a Laurea in Computer Science at the University of Udine and a PhD in Computer Science at the University *La Sapienza* of Rome. He is a researcher in Computer Science at the University of L'Aquila. His research interests include automated deduction, computational logic, theory-based reasoning, logic programming, computable set theory, relation algebras. He has been member of program and organizing committees of national and international conferences.

Enrico Pontelli received his Laurea from the University of Udine, a Master from the University of Houston, and a Ph.D. in Computer Science from New Mexico State University. He is a full professor of Computer Science at New Mexico State University. His research interests are in the area of logic and constraint programming, parallel processing, bioinformatics, and assistive technologies. He is member of the Executive Committee of the Association for Logic Programming and Editor-in-Chief of the ALP newsletter. He is also member of the ACM. He has served in program committees of various conferences on declarative programming and artificial intelligence.

Photograph.

Agostino: <http://www.dimi.uniud.it/dovier/FOTO/dovier.jpg>

Andrea: <http://www.di.univaq.it/~formisano/andy2BWb.jpg>

Enrico: <http://www.cs.nmsu.edu/~epontell/officel.jpg>