

---

# A comparison of CLP(FD) and ASP solutions to NP-complete problems

Agostino Dovier

Univ. di Udine

Dip. di Matematica e Informatica.

Andrea Formisano

Univ. di L'Aquila

Dip. di Informatica.

Enrico Pontelli

New Mexico State University

Dept. of Computer Science

This work is partially supported by:

**GNCS2005** project on constraints and their applications,  
and by **NSF** grant EIA-0220590.

# Motivations (1)

---

- NP-complete (decision) problems and the corresponding minimization problems are naturally encoded using *Constraint Programming on Finite Domains*
- Programming style is known as *constrain and generate*
- Solution's search is done using *prop labeling trees* that interleave constraints *propagation* with solutions attempts for one variable (labeling)
- Several built-in primitive constraints with clever propagation algorithms are available.

## Motivations (2)

---

- Determining the semantics of normal (i.e., with negation) non-stratified logic programs has been an open research issue for years.
- The *stable model semantics*, proposed by Gelfond and Lifschitz have widely been accepted by the community.
- A normal program always admits Herbrand models, but it can have 0, 1, or more stable models.
- Tools for computing the stable models from a *ground, finite* general program have been developed (e.g., Smodels, DLV, ASSAT, Cmodels).

# Motivations (3)

---

- Stable models are also known as *answer sets*.
- *Answer Set Programming* is a declarative programming paradigm where the problem is encoded so that the answer sets are the expected solutions.
- It is well-known that determining whether a general program admits an answer set is NP-complete.
- As a consequence, ASP can be naturally used for a declarative encoding of NP-complete problems.
- Solutions are found by Smodels, Cmodels, etc. using different techniques (resolution techniques, partial or full look ahead, translation to SAT problems, . . .)

# Objective of this work

---

- Comparing Constraint *Logic* Programming on finite domains (the constraint-based language closest to ASP) and ASP encoding of some NP-complete or combinatorial problems.
- We compare both the encoding style and the execution time.
- Our overall choice is to keep the problem encoding as declarative and simple as possible.
- We try to exclude programming ability/inability from the test.

# An example: Schur Numbers

- A set  $S \subseteq \mathbb{N}$  is *sum-free* if the intersection of  $S$  and the set  $S + S = \{x + y : x \in S, y \in S\}$  is empty.
- The *Schur number*  $S(P)$  is the largest integer  $n$  for which the interval  $[1..n]$  can be partitioned in  $P$  sum-free sets.
- For instance,  $\{1, 2, 3, 4\} \Rightarrow S_1 = \{1, 4\}$  and  $S_2 = \{2, 3\}$ .
- $\{1, 2, 3, 4, 5\}$  originates at least 3 sum-free subsets.
  - 1 and 2 must stay in different sets.
  - Similarly for 2 and 4.
  - Using two sets, the unique partial sol. is:  $\{1, 4\}, \{2\}$ .
  - Neither 3 nor 5 cannot stay in the first set.
  - And,  $\{2, 3, 5\}$  is not sum-free.
- $S(1) = 1, S(2) = 4, S(3) = 13, S(4) = 44$ . The best known bounds for  $S(5)$  are  $160 \leq S(5) \leq 315$ .
- We focus on the decision problem  $S(P) \geq N$ .

# Schur (Abstract) Formalization

- The problem is  $S(P) \geq N$ ?
- We look for a function
  - $B : \{1, \dots, N\} \longrightarrow \{1, \dots, P\}$such that:
  - $(\forall I \in \{1, \dots, N\})$   
 $(\forall J \in \{I, \dots, N\})$   
if  $B(I) = B(J)$  then  $B(I + J) \neq B(I)$
- In *CLP(FD)* we introduce a list of constrained variables  $[B_1, \dots, B_N]$  ranging on  $1, \dots, P$ .
- In ASP we define a predicate `inpart(Number, Block)` implementing the function  $B$  from numbers  $1, \dots, N$  to blocks  $1, \dots, P$ .

# Schur Numbers in CLP(FD)

```
schur(N,P) :- length(List,N),
              domain(List,1,P),   %%% List = [B1,...,BN]
              constraints(List,N),
              labeling([ff],List).

constraints(List, N) :- recursion(List,1,1,N).
recursion(_ ,I,_,N):- I>N, !.
recursion(List,I,J,N):- I+J>N, !, I1 is I+1, recursion(List,I1,1,N).
recursion(List,I,J,N):- I>J, !, J1 is J+1, recursion(List,I,J1,N).
recursion(List,I,J,N):- K is I+J, J1 is J+1,
                        nth(I,List,BI), nth(J,List,BJ), nth(K,List,BK),
                        (BI #\= BJ) #=> (BK #\= BI),          (*)
                        recursion(List,I,J1,N).
```

Replacing (\*) with

$$BK \neq BI + 10 * (BI - BJ)$$

there is a speed up. Which one must be used for tests?

# Schur Numbers in ASP

Domain predicates:

```
number(1..n).                part(1..p).
```

Each number in  $1..n$  belongs to exactly one block of the partition.

```
1 { inpart(X,Block) : part(Block) } 1 :- number(X).
```

Alternatively, using standard clauses (it runs slower):

```
not_inpart(X,B) :- inpart(X,B1),B!=B1,number(X),part(B;B1).  
inpart(X,B) :- not not_inpart(X,B),number(X),part(B).
```

It cannot be that  $X$  and  $Y$  are in the same block  $B$  of  $X + Y$ .

```
:- inpart(X,B),inpart(Y,B),inpart(X+Y,B),X<=Y,number(X;Y), part(B).
```

( $X \leq Y$  removes redundancy)

## Schur Numbers in ASP (2)

- For (39, 4) it found a sol. in 44.7s, rules 3358.
- To avoid empty blocks, we could state that:  
`1 { inpart(X,P) : number(X) } n :- part(P).`
- $\Rightarrow$  45.5s, (3374). Otherwise, we could ask:  
`nonempty(X,P) :- inpart(Y,P), Y !=X, number(X;Y), part(P).`  
`:- inpart(X,P), P1<P, not nonempty(X,P1), number(X), part(P;P1).`
- $\Rightarrow$  28.4s, (9520) The following asymmetric version is usually employed in ASP (it forces `inpart(1,1), inpart(2,2)`)  
`nonempty(X,P) :- inpart(Y,P), Y<X, number(X;Y), part(P).`  
`:- inpart(X,P), P1<P, not nonempty(X,P1), number(X), part(P;P1).`
- $\Rightarrow$  0.45s (6556) This is the one we tested.

# Schur: Computational Results

$\langle P, N \rangle$		SModels	CModels	CLP(FD)	CLP(FD) (*)
$\langle 4, 44 \rangle$	Y	0.29	3.35	4.18	2.0
$\langle 4, 45 \rangle$	N	510.01	891.66	—	66.4
$\langle 4, 46 \rangle$	N	561.80	813.34	—	73.6
$\langle 4, 47 \rangle$	N	767.80	791.57	—	77.6
$\langle 4, 48 \rangle$	N	978.84	805.33	—	89.7
$\langle 4, 49 \rangle$	N	1258.57	678.19	—	96.0
$\langle 4, 50 \rangle$	N	1680.34	890.05	—	99.7
$\langle 5, 105 \rangle$	Y	—	27.88	0.37	4.5
$\langle 5, 106 \rangle$	Y	—	38.55	0.40	5.4
$\langle 5, 107 \rangle$	Y	—	5.07	0.44	5.8
$\langle 5, 108 \rangle$	Y	—	2.80	0.43	6.4
$\langle 5, 109 \rangle$	Y	—	13.97	0.44	35.0
$\langle 5, 110 \rangle$	Y	—	33.12	54.71	39.5
$\langle 5, 111 \rangle$	Y	—	0.52	56.72	717.5
$\langle 5, 112 \rangle$	Y	—	0.54	58.44	708.5
$\langle 5, 113 \rangle$	Y	—	0.54	207.46	724.9
$\langle 5, 114 \rangle$	Y	—	11.63	1032.43	1043.7
$\langle 5, 115 \rangle$	Y	—	82.13	1069.54	1081.3
$\langle 5, 116 \rangle$	Y	—	60.53	1108.02	1100.0
$\langle 5, 117 \rangle$	Y	—	761.11	1150.25	1144.3

(\*) We have forced  $X_1 = 1, X_2 = 2$ , and used the  $\neq$  constraint.

# (Protein) Structure Prediction

- Given  $S = s_1 \cdots s_n$ ,  $s_i \in \{h, p\}$ , the *2D, PSP problem* is the problem of finding a mapping (*folding*)

- $\omega : \{1, \dots, n\} \longrightarrow \mathbb{N}^2$

such that

- $(\forall i \in [1, n - 1]) \text{next}(\omega(i), \omega(i + 1))$ , and
- $(\forall i, j \in [1, n])(i \neq j \rightarrow \omega(i) \neq \omega(j))$

minimizing the (energy) function:

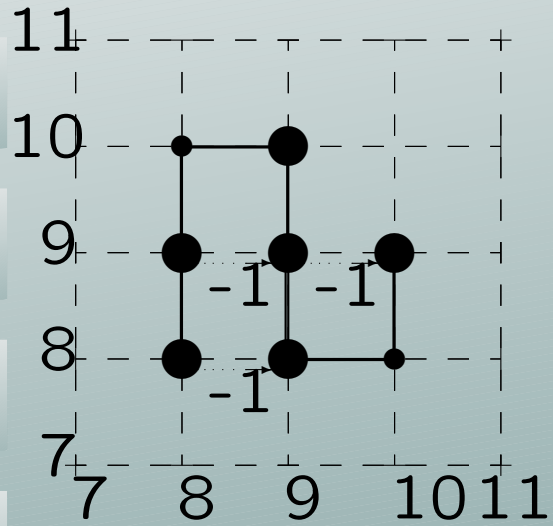
$$\sum_{\substack{1 \leq i \leq n - 2 \\ i + 2 \leq j \leq n}} \text{Pot}(s_i, s_j) \cdot \text{next}(\omega(i), \omega(j))$$

where:

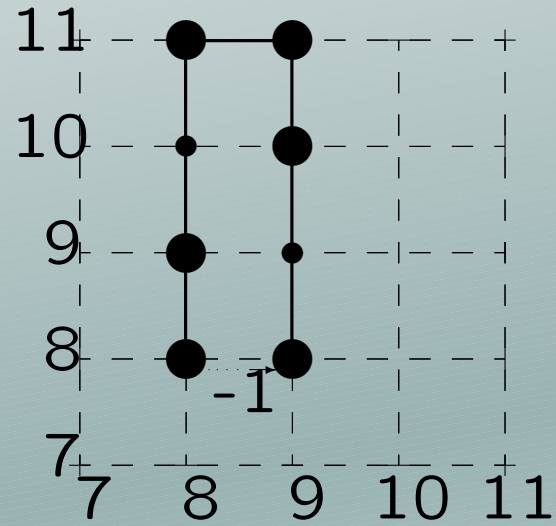
- $\text{Pot}(s_i, s_j) \in \{0, -1\}$  and
- $\text{Pot} = -1$  if and only if  $s_i = s_j = h$ .

- $\text{next}(\langle X_1, Y_1 \rangle, \langle X_2, Y_2 \rangle)$  holds iff  $|X_1 - X_2| + |Y_1 - Y_2| = 1$ .
- W.l.o.g., let  $\omega(1) = \langle n, n \rangle$  and  $\omega(2) = \langle n, n + 1 \rangle$ .

# Two foldings



Value: -3



Value: -1

# CLP Encoding

- A natural constraint + generate code

```
pf(Primary, Tertiary) :-    %%% Primary = [a1,...,aN], ai in {h,p}
    constrain(Primary,Tertiary,Energy),
    labeling([ff,minimize(Energy)],Tertiary).
constrain(Primary,Tertiary,Energy) :-
    length(Primary,N),
    M is 2*N, M1 is M - 1,
    length(Tertiary,M),      %%% Tertiary = [X1,Y1,...,XN,YN]
    set_domains(Tertiary,N), %%% Xi, Yi in [N-sqrt(N),N+sqrt(N)]
    starting_point(Tertiary,N), %%% X1=Y1=X2=N, Y1=N+1
    avoid_loops(Tertiary),   %%% (Xi,Yi) != (Xj,Yj)
    next_constraints(Tertiary), %%% |(X_i-X_{i+1}) + (Y_i - Y_{i+1})| = 1
    energy_constraint(Primary,Tertiary,Energy).
```

- For details, see, e.g., my tutorial in CILC 2004.

# ASP encoding

- Input Representation:

```
prot(1,h). prot(2,p). prot(3,p). prot(4,h). prot(5,p).  
prot(6,p). prot(7,h). prot(8,p). prot(9,p). prot(10,h).
```

- Domains

```
(1) size(10).      %%% size(N) where N is input length  
(2) range(7..13). %%% [ N-sqrt{N}, N+sqrt{N} ]
```

- We look for a function  $\text{sol} : \{1, \dots, N\} \rightarrow \mathbb{N}^2$ .

```
(3) sol(1,N,N)    :- size(N). %%% starting  
(4) sol(2,N,N+1) :- size(N). %%% points  
(5) 1 { sol(I,X,Y) : range(X;Y) } 1 :- prot(I,Amino).
```

- No loop constraint:

```
(6) :- prot(I1,A1), prot(I2,A2), neq(I1,I2),  
      sol(I1,X,Y), sol(I2,X,Y), range(X;Y).
```

## ASP encoding (2)

- Two aminoacids are next

```
(7) :- prot(I1,A1), prot(I2,A2), I2>1,  
      eq(I1,I2-1), not next(I1,I2).
```

```
(8) next(I1,I2) :- prot(I1,A1), prot(I2,A2), I1<I2,  
      sol(I1,X1,Y1), sol(I2,X2,Y2), range(X1;Y1;X2;Y2),  
      1==abs(Y1-Y2)+abs(X2-X1).
```

- We define the energy and maximize it

```
(9) energy_pair(I1,I2) :- prot(I1,h), prot(I2,h),  
      next(I1,I2), I1+2<I2, 1==(I2-I1) mod 2.
```

```
(10) seq_proteins(I1,I2) :- prot(I1,A1), prot(I2,A2),  
      I1+2<I2, 1==(I2-I1) mod 2.
```

```
(11) maximize{ energy_pair(I1,I2) : seq_proteins(I1,I2) }.
```

# PSP: Computational Results

Instance			Optimization problem		Decision problem		
Input	N	Min	CLP(FD)	SModels	CLP(FD)	SModels	CModels
$h^{10}$	10	-4	0.14	0.91	0.01	0.65	0.69
$h^{15}$	15	-8	1.93	13.21	0.04	2.84	2.69
$h^{20}$	20	-12	201.58	1982.44	0.29	45.63	40.70
$h^{25}$	25	-16	25576.38	—	817.71	3181.78	1165.26
$(hpp)^3h$	10	-4	0.01	0.66	0.01	0.42	0.49
$(hpp)^5h$	16	-6	0.32	26.95	0.22	22.46	16.58
$(hpp)^7h$	22	-6	62.69	1303.13	12.75	35.96	161.25
$(hpp)^9h$	28	-9	6758.45	—	1955.28	3369.78	1217.34

# General observations

---

- CLP(FD) programs are based on lists of constrained variables
- In CLP(FD), most of the effort for encoding the problem is spent writing (simple) recursive procedures on these lists.
- ASP programs are instead based on predicate definitions
- ASP constraints (i.e., goals) allows a more compact encoding of recursion.
- All programs and results used can be found here:  
`www.di.univaq.it/~formisano/CLPASP`
- We have used SICStus clpfd (faster than ECLiPSe ic in our tests) and Smodels and Cmodels ASP solvers.

# Running Time Comparison

- We can summarize the main points of our tests as follows:
- Save some (small) cases, Smodels is slower than Cmodels. The competition is CLP(FD) vs ASP-Cmodels
- 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)

	Coloring	Hamilton	Schur	PF	Planning	Knapsack
CLP(FD)	+	++	+	+	+	+
ASP CModels	++	+	++	-	+	-

# Conclusions

---

- This is a *preliminary* work
- However, we executed thousands of tests, and we have some strong indications.
- *Short term*: we wish to extend tests to optimized versions of the programs and test other solvers.
- *Long term*: developing an integrated solver