Efficient ASP Techniques for Solving Hard Problems

Carmine Dodaro
DIBRIS, University of Genoa

Arcavacata di Rende, 2-5-6-7 February 2018

Pigeon Problem

Description

Given a set of n pigeons and m holes, assign each pigeon to exactly one hole such that two pigeons do not share the same hole.

Pigeon Problem: Natural Encoding

```
{inHole(X,H) : hole(H)} = 1 :- pigeon(X).
:- inHole(X,H), inHole(Y,H), X < Y.
```

Pigeon Problem: Optimized Encoding

```
{inHole(X,H) : hole(H)} = 1 :- pigeon(X).
:- inHole(X,H), inHole(Y,H), X < Y.
:- #count{X:pigeon(X)}=P,#count{X:hole(X)}=H,P>H.
```

Instantiation of Large Aggregates (1)

```
c(1). c(2). ... c(9999). c(10000).
a(X) | b(X) :- c(X).
:- #count{X:a(X)}=N1, #count{X:b(X)}=N2, N1=N2.
```

Instantiation of Large Aggregates (1)

```
c(1). c(2). ... c(9999). c(10000).
a(X) | b(X) :- c(X).
:- #count{X:a(X)}=N1, #count{X:b(X)}=N2, N1=N2.
```

Such aggregates are quite difficult to be evaluated

Instantiation of Large Aggregates (2)

- A possible solution is to explore different ad-hoc strategies for the problem
- What is the simplest strategy?

Instantiation of Large Aggregates (2)

- A possible solution is to explore different ad-hoc strategies for the problem
- What is the simplest strategy? In this simple case the number of true “a” must be different from 5000!

A general solution

Ideas?

A general solution

Ideas?

```
c(1). c(2). ... c(9999). c(10000).
a(X) | b(X) :- c(X).
:- #sum{1,X : a(X); -1,Y : b(Y)} = 0.
```

Nurse Scheduling Problem (NSP)

Goal

Generation of schedules for nurses consisting of working and rest days over a predetermined period of time

Motivation

A proper solution to NSP

- Guarantees the high level of quality of health care
- Improves the degree of satisfaction of nurses
- The recruitment of qualified personnel

Problem Description

Requirements

- Planning period: one year
- Three working shifts: morning (7 A.M. – 2 P.M.), afternoon (2 P.M. – 9 P.M.), night (9 P.M. – 7 A.M.)
- Coverage: min and max number of nurses in each shift
- Workload: min and max number of working hours per year
- Vacation: 30 days per year
- The starting time of a shift must be at least 24 hours later than the starting time of the previous shift
- Each nurse has at least two rest days each fourteen days
- After two consecutive nights there is one special rest day
- Balance: Morning, afternoon and night shifts assigned to every nurse should range over a set of acceptable values

ASP Encoding: Define Search Space

Definition of Shifts

shift(1, morning, 7). shift(2, afternoon, 7).
shift(3, night, 10). shift(4, specialrest, 0).
shift(5, rest, 0). shift(6, vacation, 0).

```
% Choose an assignment for each day and for each
   nurse.
{assign(N,S,D):shift(S,Name,H)}=1 :- nurse(N),
   day(D).
```

ASP Encoding: Constraints (1)

```
% Coverage: Limits to nurses that must be present
    for each shift.
:- day(D), #count{N:assign(N,S,D)}>Max,
   nurseLimits(S,Min,Max).
:- day(D), #count{N:assign(N,S,D)}<Min,
   nurseLimits(S,Min,Max).

% Workload: Min and Max working hours per year.
:- nurse(N), #sum{H,D:assign(N,S,D),shift(S,H)} >
   Max, workLimits(Min,Max).
:- nurse(N), #sum{H,D:assign(N,S,D),shift(S,H)} <
   Min, workLimits(Min,Max).

% Holidays: Exactly 30 days of vacation (ID 6)
:- nurse(N), #count{D : assign(N,6,D)} != 30.
```

ASP Encoding: Constraints (2)

```
% Each nurse cannot work twice in 24 hours.
:- nurse(N), assign(N,T1,D), assign(N,T2,D+1),
   T2<T1, T1<=3.

% At least 2 rest days each 14 days.
:- nurse(N),day(D),days(DAYS),D<=DAYS-13,
#count{D1:assign(N,5,D1), D1>=D, D1<=D+13}<2.

% Special rest day after two nights (IDs 3 and 4
   are night and special rest).
:- not assign(N,4,D), assign(N,3,D-2),
   assign(N,3,D-1).
:- assign(N,4,D), not assign(N,3,D-2).
:- assign(N,4,D), not assign(N,3,D-1).
```

ASP Encoding: Constraints (3)

```
% Balance: Morning, afternoon and night shifts
    assigned to every nurse should range over a
    set of acceptable values.
:- nurse(N), #count{D : assign(N,S,D)} > Max,
    dayLimits(S,Min,Max).
:- nurse(N), #count{D : assign(N,S,D)} < Min,
    dayLimits(S,Min,Max).
```

Inefficiencies of the encoding (1)

Consider only the following portion of program:

```
% Workload: Min and Max working hours per year.
:- nurse(N), #sum{H,D:assign(N,S,D),shift(S,H)} >
   Max, workLimits(Min,Max).
:- nurse(N), #sum{H,D:assign(N,S,D),shift(S,H)} <
   Min, workLimits(Min,Max).

% Balance: Morning, afternoon and night shifts
   assigned to every nurse should range over a
   set of acceptable values.
:- nurse(N), #count{D : assign(N,S,D)} > Max,
   dayLimits(S,Min,Max).
:- nurse(N), #count{D : assign(N,S,D)} < Min,
   dayLimits(S,Min,Max).
```

Inefficiencies of the encoding (2)

Assume the following values for the constraints:

- Assume 3 shifts:
  shift(1,morning,7) shift(2, afternoon, 7) shift(3, night, 10)
- Workload: min and max number of working hours per year, 1687 and 1692
- Balance: a nurse must be assigned to at least 74 and at most 82 mornings, 74 to 82 afternoons and 58 to 61 nights

Assume the following values for the constraints:

- Assume 3 shifts:

  shift(1,morning,7) shift(2, afternoon, 7) shift(3, night, 10)
- Workload: min and max number of working hours per year, 1687 and 1692
- Balance: a nurse must be assigned to at least 74 and at most 82 mornings, 74 to 82 afternoons and 58 to 61 nights

N	$M+A$																
	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164
58	1616	1623	1630	1637	1644	1651	1658	1665	1672	1679	1686	1693	1700	1707	1714	1721	1728
59	1626	1633	1640	1647	1654	1661	1668	1675	1682	1689	1696	1703	1710	1717	1724	1731	1738
60	1636	1643	1650	1657	1664	1671	1678	1685	1692	1699	1706	1713	1720	1727	1734	1741	1748
61	1646	1653	1660	1667	1674	1681	1688	1695	1702	1709	1716	1723	1730	1737	1744	1751	1758

- Number of working hours assigned to nurse n, that is, $7 \times (M + A) + 10 \times N$
- Admissible values ([1687..1692]) are emphasized in red

Possible Optimization: Intuition

```
admissible(N,M+A) :-
  dayLimits(1,MinM,MaxM), M >= MinM, M <= MaxM,
  dayLimits(2,MinA,MaxA), A >= MinA, A <= MaxA,
  dayLimits(3,MinN,MaxN), N >= MinN, N <= MaxN,
  V=7*(M+A)+10*N,workLimits(MinW, MaxW),
      MinW<=V<=MaxW.

values(N,S,V):- nurse(N), dayLimits(S,Min,Max),
  #count{D:assign(N,S,D)}=V, V >= Min, V <= Max.

%V1,V2,V3 represent the number of mornings,
    afternoons, and nights
valid(Nu) :- nurse(Nu), admissible(N,M+A),
  values(Nu,1,M),values(Nu,2,A),values(Nu,3,N).

:- nurse(N), not valid(N).
```

Impact

Solving time (in seconds) on five instances (a dash means not solved in 1 hour)

Solver	**Nurses**				
	10	**20**	**41**	**82**	**164**
CLINGO (SIMPLE ENC)	155	117	738	1486	2987
CLINGO (OPTIMIZED ENC)	4	9	70	351	1291
GLUCOSE (SAT ENC)	-	-	-	-	-
GLUCOSE (SAT ENC)	-	-	-	-	-
CLASP (SAT ENC)	-	-	-	-	-
GUROBI (ILP ENC)	62	172	1018	-	-

Beyond the Encodings

- What should we do when encodings cannot be (easily) improved and our ASP system is still inefficient?

Beyond the Encodings

- What should we do when encodings cannot be (easily) improved and our ASP system is still inefficient?

  Try different strategies of the solvers

Beyond the Encodings

- What should we do when encodings cannot be (easily) improved and our ASP system is still inefficient?

  Try different strategies of the solvers

  Try a different solver

Beyond the Encodings

- What should we do when encodings cannot be (easily) improved and our ASP system is still inefficient?

  Try different strategies of the solvers

  Try a different solver

  Try to improve the branching heuristics of solvers

Beyond the Encodings

- What should we do when encodings cannot be (easily) improved and our ASP system is still inefficient?

  Try different strategies of the solvers

  Try a different solver

  Try to improve the branching heuristics of solvers

  Implement ad-hoc propagators

Beyond the Encodings

- What should we do when encodings cannot be (easily) improved and our ASP system is still inefficient?

  Try different strategies of the solvers

  Try a different solver

  Try to improve the branching heuristics of solvers

  Implement ad-hoc propagators

Try Different Options

- ASP systems have several options (usually $--$help shows them)
- Different options can have a huge impact on the performance
- Let's see a practical example!

Heuristic Example: PUP and CCP

- Partner Units Problem (PUP)
- Combined Configuration Problem (CCP)
- Two hard problems proposed by Siemens
- Several efficient encodings have been proposed but still performance are not good

Heuristic Example

Partner Units Problem

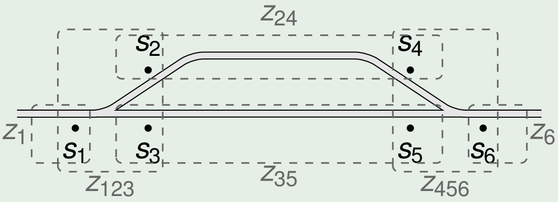
- Railway tracks are equipped with sensors registering wagons
- Sensors are organized in safety zones
- A set of control units enforces safety requirements on connected zones and sensors

PUP instance

A PUP instance

- A layout of sensors S and zones Z represented as an undirected bipartite graph G=(S ∪ Z, E)
- A set of units U
- The maximum number of sensors/zones connected to a unit UCAP
- The maximum number of inter-unit connections IUCAP

Example (U={u_1, u_2, u_3}, UCAP = IUCAP = 2)



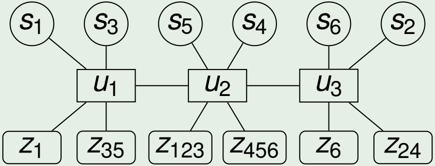
PUP solution

Solution

A solution is an assignment of zones and sensors to units and an interconnection of units, such that

- Every unit is connected to at most UCAP sensors and at most UCAP zones
- Every unit is connected to at most IUCAP partner units
- If a sensor s is part of a zone z, then s must be connected to the same or a partner unit of the unit connected to z

Example (U={u_1, u_2, u_3}, UCAP = IUCAP = 2)



Heuristic Example: Definition

Combined Configuration Problem

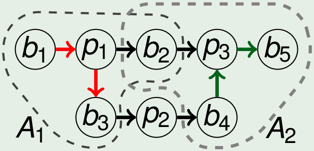
- Abstracts a number of real-world problems including railway interlocking systems, safety automation, and resource distribution
- Occurs in practical applications at Siemens, where a complex problem is composed of a set of subproblems

CCP instance

A CCP instance

- A directed acyclic graph $G = (V, E)$, where each vertex has a type (e.g., b, p) with an associated size (e.g., 1, 3)
- Two disjoint paths P_1 (red arrows) and P_2 (green arrows)
- A set of safety areas and their border elements
- The max number M of assigned border elements per area
- C colors, B bins per color, K the capacity of each bin

Example (B=2, M=C=K=3)



CCP solution

Solution

Assignment of colors to vertices, vertices to bins and border elements to areas such that the following subproblems are solved

(P1) Coloring: Every vertex must have exactly one color

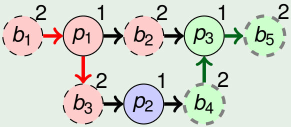
(P2) Bin-Packing: For each set $V_c \subseteq V$ of color c, assign each vertex in V_c to exactly one bin s.t. for each bin the sum of vertex sizes is not greater than K, and at most B bins are used

(P3) Disjoint Paths: Vertices in different paths must not share a color

(P4) Matching: For each area A assign a set of border elements, such that all border elements have the same color and A has at most M border elements. Additionally, each border element must be assigned to exactly one area

(P5) Connectedness: Two vertices of the same color must be connected via a path that comprises only vertices of this color

Example (B=2, M=C=K=3)



PUP and CCP

- Problems have been solved in ASP using custom branching heuristics (sometimes called domain-specific heuristics)
- Heuristics have been implemented on top of the ASP solver WASP
- WASP with domain heuristics was able to solve all the instances

Packing Problem

Definition

The Packing Problem is related to a class of problems in which one has to pack objects together in a given container. The problem submitted to 3rd ASP Competition was the packing of squares of possibly different sizes in a rectangular space and without the possibility of performing rotations. The encoding follows the typical guess-and-check structure, where positions of squares are guessed and some constraints check whether the guessed solution is an answer set.

Custom propagators: Packing Problem

Packing Problem

- Problem submitted to the ASPCOMP 2011
- Grounding bottleneck $\rightarrow$ ‘Non groundable” since (ASPCOMP 2014)
- Few non ground constraints depending *on the size of the grid*

Problem statement

They are given

- a rectangular area of a known dimension *n*
- a set of squares of size *s*

Pack all the squares into the rectangular area s.t. no squares overlap

Custom propagators: Packing Problem

Packing Problem

- Problem submitted to the ASPCOMP 2011
- Grounding bottleneck $\rightarrow$ 'Non groundable" since (ASPCOMP 2014)
- Few non ground constraints depending *on the size of the grid*

Problematic constraints

% The same square cannot be assigned to different positions

$:- pos(I, X, Y), pos(I, X_1, Y_1), X_1 \neq X$

$:- pos(I, X, Y), pos(I, X_1, Y_1), Y_1 \neq Y$

% Squares cannot overlap

$:- pos(I_1, X_1, Y_1),\ square(I_1, D_1),\ pos(I_2, X_2, Y_2),\ square(I_2, D_2), I1 \neq I2,$
$W1 = X1 + D1, H1 = Y1 + D1,\ X2 \geq X1,\ X2 < W1,\ Y2 \geq Y1,\ Y2 < H1.$

Custom propagators: Packing Problem

Packing Problem

- Problem submitted to the ASPCOMP 2011
- Grounding bottleneck $\rightarrow$ ‘Non groundable” since (ASPCOMP 2014)
- Few non ground constraints depending *on the size of the grid*

Problematic constraints

% The same square cannot be assigned to different positions

$:\!-\, pos(I, X, Y), pos(I, X_1, Y_1), X_1 \neq X$

$:\!-\, pos(I, X, Y), pos(I, X_1, Y_1), Y_1 \neq Y$

% Squares cannot overlap

$:\!-\, pos(I_1, X_1, Y_1),\ square(I_1, D_1),\ pos(I_2, X_2, Y_2),\ square(I_2, D_2), I1 \neq I2,$
$W1 = X1 + D1, H1 = Y1 + D1,\ X2 \geq X1,\ X2 < W1,\ Y2 \geq Y1,\ Y2 < H1.$

$\longrightarrow$ idea: replace such constraints by means of custom propagators

A practical exercise for you

Use an ASP encoding to answer the problem at `https://web.stanford.edu/~laurik/fsmbook/examples/Einstein'sPuzzle.html`

...The end

Thank you for attending the course!