Solving Challenging Grid Puzzles with Answer Set Programming

Merve Çaylı, Ayşe Gül Karatop, Emrah Kavlak, Hakan Kaynar
Ferhan Türe, and Esra Erdem

Sabancı University
Faculty of Engineering and Natural Sciences
Orhanlı, Tuzla, Istanbul 34956, TURKEY

Abstract. This note studies four challenging grid puzzles, Nurikabe, Heyawake, Masyu, Bag Puzzle, interesting for answer set programming both from the point of view of representation and from the point of view of computation.

1 Introduction

Puzzles like Blocks World, Sudoku, Cross Sum, and Sokoban have been studied in various areas of AI (e.g., knowledge representation, planning, constraint programming, answer set programming) as toy problems, and these studies have led to interesting insights and useful approaches for solving them as well as some real-life problems. On the other hand, there are many other challenging puzzles that have not been well-studied, but could be useful to investigate some theoretical or application-oriented questions. In this note we study four such Nikoli puzzles,[1] Nurikabe, Bag Puzzle, Masyu, and Heyawake, interesting for answer set programming (ASP) [7, 8, 10] both from the point of view of representation and from the point of view of computation.

All puzzles require (several versions of) "connectedness", and some require the presence of a "single loop"; thus their straightforward representations require transitive closure. Since these puzzles include constraints involving "counting", they need aggregates as well. Such requirements make it challenging to represent these puzzles in other declarative formalisms, while it is convenient to represent them in ASP. Since straightforward representations of these puzzles are not tight, and computation of solutions by some answer set solvers (like CMODELS[2] and CLASP[3]) involves generation of loop formulas (as many as 861656), these puzzles form an interesting suite of benchmarks for answer set solvers, in particular, to evaluate algorithms for handling loop formulas. All problem instances used in our experiments are available at `http://people.sabanciuniv.edu/~esraerdem/ASP-benchmarks/puzzles.html`.

In this note, we represent each puzzle in the language of LPARSE, show how their instances can be solved using the answer set solver SMODELS,[4] compare its computational efficiency on these problems with that of CMODELS and CLASP, and conclude with a discussion of the experimental results and future work.

[1] `http://www.nikoli.co.jp/en/.`
[2] `http://www.cs.utexas.edu/users/tag/cmodels.html.`
[3] `http://www.cs.uni-potsdam.de/clasp/.`
[4] `http://www.tcs.hut.fi/Software/smodels/.`

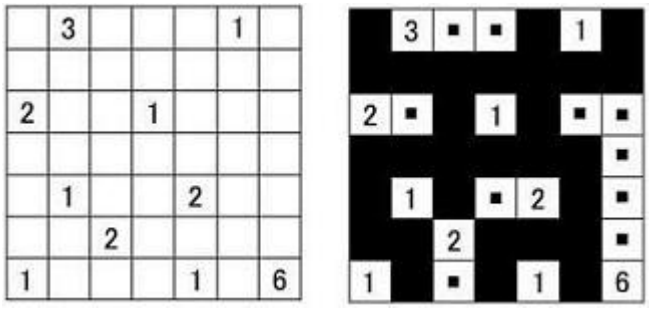

Fig. 1. A sample Nurikabe puzzle with its solution.

```
numbered(1,1,1). numbered(1,5,1). numbered(1,7,6). numbered(2,3,2).
numbered(3,2,1). numbered(3,5,2). numbered(5,1,2). numbered(5,4,1).
numbered(7,2,3). numbered(7,6,1).
```

Fig. 2. The puzzle in Fig. 1 described in the language of LPARSE.

2 Nurikabe

Nurikabe (also called Cell Structure and Islands in the Stream) is played on a rectangular grid. Initially all grid cells are white and some of them are numbered, like in Fig. 1. The goal is to paint some of the grid cells black, like in Fig. 1, according to the rules below, so that the black cells (water) form the "nurikabe" and the white cells (land) form "islands". Here are the rules of Nurikabe:

N1 Numbered cells should remain white.
N2 Every white cell belongs to an island.
N3 Each island must contain exactly one numbered cell.
N4 Each island should contain the same number of white cells as the number it contains.
N5 In every island, white cells should be orthogonally connected.
N6 Two islands can not be connected.
N7 All black cells should be connected orthogonally.
N8 No subset of black cells forms a 2×2 square.

Deciding whether there is a solution to Nurikabe is NP-complete [9].

We describe Nurikabe in the language of LPARSE as follows.

Input and Output Suppose that we are given a Nurikabe puzzle with a rectangular grid, with n rows and m columns, containing some numbered cells. The rows and the columns are described by atoms of the forms `row(X)` and `col(X)` respectively. The numbered cells are described by atoms of the form `numbered(X,Y,Z)` ("grid cell `(X,Y)`, located at column `X` and row `Y`, contains number `Z`, and thus belongs to an island of size `Z`"). For instance, the puzzle in Fig. 1 is described in the language of LPARSE as in Fig. 2. Solutions of Nurikabe puzzles can be characterized by atoms of the form `black(X,Y)` ("grid cell `(X,Y)` is painted black"). For instance, for the puzzle in Fig. 1, SMODELS computes the solution in that figure, as in Fig. 3.

```
black(2,1) black(3,1) black(4,1) black(6,1) black(7,1) black(1,2)
black(2,2) black(4,2) black(6,2) black(3,3) black(4,3) black(5,3)
black(6,3) black(1,4) black(2,4) black(4,4) black(6,4) black(2,5)
black(4,5) black(5,5) black(6,5) black(7,5) black(1,6) black(2,6)
black(3,6) black(4,6) black(6,6) black(6,7) black(7,7)
```

Fig. 3. The solution of the puzzle in Fig. 1 computed by SMODELS.

Main Program To compute a solution like in Fig. 3, in addition to the specific puzzle description, as in Fig. 2, we need to describe Nurikabe in the language of LPARSE, e.g., as in Fig. 4. In this formulation first a set of cells that would remain white is generated by a choice rule so that the rest are painted black. Then the conditions N1–N8 are guaranteed by some constraints, some of which require further auxiliary definitions. For instance, to guarantee the connectedness of the nodes of the same color, first we define the adjacency of two nodes in the grid (by `adj`), and then introduce a definition for orthogonal connectedness of the cells of the same color (by `connected`).

3 Heyawake

Heyawake is played on a rectangular grid whose cells are white or numbered, like in Nurikabe. Furthermore, different from Nurikabe, the grid is divided into rooms, as shown in Fig. 5. The goal is, like in Nurikabe, to paint some cells black (Fig. 5), but obeying a different set of rules:

H1 Any two black cells should not be horizontally or vertically adjacent to each other.
H2 All white cells should be interconnected.
H3 Each number contained in a room indicates how many cells in that room should be painted black.
H4 If a room does not contain any numbers, it may contain any number of black cells.
H5 No straight path connecting two white cells passes through more than two rooms.

We describe Heyawake in the language of LPARSE as follows.

Input and Output Suppose that we are given a Heyawake puzzle with a rectangular grid, with n rows and m columns, divided into r rectangular rooms, each containing at most one numbered cell. Suppose also that the rooms are enumerated, so each room has a label. We describe that a room with corners `(X1,Y1)`, `(X1,Y2)`, `(X2,Y1)`, and `(X2,Y2)` is labeled `A`, with atoms of the form `room(A,X1,Y1,X2,Y2)`. We describe that Room `A` contains a cell numbered `N` with an atom of the form `has(A,N)`. If the room does not contain any numbered cell then `N` is set to -1. For instance, the input file describing the Heyawake puzzle in Fig. 5 is presented in Fig. 6. Solutions to Heyawake puzzles can be characterized by atoms of the form `black(X,Y)` ("grid cell `(X,Y)`, with column `X` and row `Y`, is painted black"), like in Nurikabe.

Main Program We can describe Heyawake in the language of LPARSE as in Fig.s 7 and 8. In this formulation, for each room, a set of cells that would be painted black is generated, taking also into account the number contained in the room. After that the

```
% File: nurikabe.lp

col(1..n).  row(1..m).  color(b;w).

% Keep some cells white (and paint the rest black)
{cell(w,X,Y)}1 :- row(X), col(Y).

% ensuring that the given constraints N1--N8 are satisfied:

% N1: Numbered cells remain white.
:- not cell(w,X,Y), numbered(X,Y,N).

% N2 & N5 : With N7, every white cell is orthogonally connected
%     to some numbered cell.
:- cell(w,X,Y), not w_connected(X,Y), row(X), col(Y).

adj(C,R,C1,R1) :- col(C;C1), row(R;R1), abs(C-C1)+abs(R-R1)==1.

connected(C,X,Y,X,Y) :- cell(C,X,Y), row(X), col(Y), color(C).
connected(C,X,Y,U,V) :- connected(C,X,Y,W,Z), adj(W,Z,U,V),
     cell(C,U,V), row(X;U;W), col(Y;V;Z), color(C).

w_connected(X,Y) :-  connected(w,X,Y,U,V),
     cell(w,X,Y), numbered(U,V,N), row(X), col(Y).

% N3 & N6: With N2, two numbered cells can not be orthogonally
%          connected.
:- numbered(X,Y,N), numbered(U,V,M), X<U, connected(w,X,Y,U,V).
:- numbered(X,Y,N), numbered(U,V,M), Y<V, connected(w,X,Y,U,V).

% N4: Each island should contain the same number of white cells
%     as the number it contains, i.e., a white cell numbered N
%     is connected to exactly N white cells:
island(X,Y) :- N {connected(w,X,Y,U,V):row(U):col(V)} N,
   numbered(X,Y,N).
:- not island(X,Y), numbered(X,Y,N).

% N7: All black cells should be connected orthogonally
:- not connected(b,X,Y,U,V), cell(b,X,Y), cell(b,U,V),
   row(X;U), col(Y;V).

cell(b,X,Y) :- not cell(w,X,Y), row(X), col(Y).

% N8: No subset of black cells forms a 2x2 square.
:- sqrBlack(X,Y), row(X), col(Y).

sqrBlack(X,Y) :- cell(b,X+1,Y), cell(b,X,Y),
     cell(b,X, Y+1), cell(b,X+1, Y+1), row(X), col(Y).

black(X,Y) :- cell(b,X,Y), row(X), col(Y).
hide. show black(_,_).
```

Fig. 4. Representing Nurikabe in the language of LPARSE

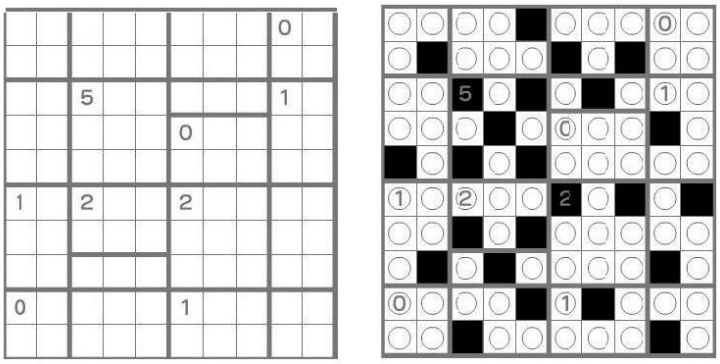

Fig. 5. A sample Heyawake puzzle and its solution.

```
room(0,1,1,2,2).room(1,1,3,2,5).room(2,6,1,8,2).room(3,3,4,5,5).
room(4,3,1,5,2).room(5,3,3,5,3).room(6,6,3,8,5).room(7,3,6,5,8).
room(8,6,6,8,7).room(9,1,6,2,8).room(10,6,8,8,8).room(13,9,6,10,8).
room(11,9,1,10,2).room(12,9,3,10,5).room(14,9,9,10,10).
room(17,1,9,2,10).room(16,3,9,5,10).room(15,6,9,8,10).
has(0,0).has(1,1).has(2,1).has(3,2).has(4,-1).has(5,-1).has(6,2).
has(7,5).has(8,0).has(9,-1).has(10,-1).has(11,-1).has(12,-1).
has(13,1).has(14,0).has(15,-1).has(16,-1).has(17,-1).
```

Fig. 6. The input file describing the Heyawake puzzle in Fig. 5.

remaining conditions H1, H2, H5 are ensured by some constraints that require auxiliary definitions. For instance, to ensure that a straight path connecting two white cells does not pass through three or more rooms, we introduce the definition of a horizontal or vertical straight path of white cells passing through 3 rooms (by `path`).

4 Masyu

Masyu (also called Pearl Puzzle) is played on a rectangular grid. Initially some grid cells contain circles; each circle is either white or black, like in Fig. 9. The goal is to draw a single, continuous line that properly passes through all circled cells, like in Fig. 9, according to the rules below:

M1 The line should form a single loop.
M2 White circles must be traveled straight through, but the loop must turn (90 degrees) in the previous and/or next cell in its path;
M3 The loop must turn (90 degrees) at black circles, but it must travel straight through the next and previous cells in its path.

Deciding whether there is a solution to Masyu is NP-complete [4].

We describe Masyu in the language of LPARSE as follows.

```
% File: heyawake.lp

col(1..m). row(1..n). num(0..r-1). segment(h;v).
room_size(N,X1,Y1,X2,Y2) :- room(A,X1,Y1,X2,Y2), has(A,N).

% H3: In each room containing a cell numbered N (N>0),
%     paint N cells black
N {black(C,R):col(C):row(R):C1<=C:C<=C2:R1<=R:R<=R2} N :-
   room_size(N,C1,R1,C2,R2), N>0.

% H4: Each room without numbered cell contains any number of
%     black cells.
{black(C,R):col(C):row(R):C1<=C:C<=C2:R1<=R:R<=R2} :-
   room_size(-1,C1,R1,C2,R2).

% H1:  No two black cells are orthogonally adjacent.
:- adj(C,R,C1,R), black(C,R), black(C1,R), C!=C1.
:- adj(C,R,C,R1), black(C,R), black(C,R1), R!=R1.

adj(C,R,C1,R1) :- col(C;C1), row(R;R1), abs(C-C1)+abs(R-R1)==1.

% H2: Nonadjacent white cells should be interconnected.
:- not connected(C,R,C1,R1), white(C,R), white(C1,R1),
   col(C;C1), row(R;R1), C!=C1, R!=R1.

white(C,R) :- not black(C,R), col(C), row(R).

% connectedness of two white cells (C,R) and (C,R1)
connected(C,R,C1,R1) :- adj(C,R,C1,R1), white(C,R), white(C1,R1).
connected(C,R,X,Y) :- connected(C1,R1,X,Y), adj(C,R,C1,R1),
   white(C,R), col(X), row(Y).
```

Fig. 7. Representing Heyawake in the language of LPARSE: Part 1.

Input and Output Suppose that we are given a Masyu puzzle with a rectangular grid, with n rows and m columns; each circle might contain a white or black circle. A grid cell `(X,Y)`, at column `X` and row `Y`, containing a black (resp. white) circle is described by an atom of the form `black(X,Y)` (resp. `white(X,Y)`). For instance, the input file describing the puzzle in Fig. 9 is presented in Fig. 10. Solutions to Masyu puzzles can be characterized by atoms of the form `line(S,X,Y)` ("if `S=h` (resp. `S=v` then the horizontal (resp. vertical) segment passing through the cells `(X,Y)` and `(X+1,Y)` (resp. `(X,Y)` and (X,Y+1)) is contained in the drawn line").

Main Program To compute a solution like in Fig. 9, in addition to the specific puzzle description, we need to describe Masyu in the language of LPARSE, e.g., as in Fig.s 11 and 12. In this formulation, first a set of line segments that would pass through balls is generated. After that the conditions M1–M3 are ensured by some constraints, some of which require auxiliary definitions. Note that this formulation includes more number of cardinality constraints compared to the previous puzzles.

```
% H5: A straight path connecting two white cells (C,R) and (X,Y)
%     does not pass through three or more rooms.
:- path(S,C,R,X,Y,3), white(C,R), white(X,Y), col(C;X), row(R;Y),
   segment(S).

% horizontal (h) and vertical (v) straight paths of white cells
% whose lengths are at most 3
path(S,C,R,C,R,1) :- white(C,R), col(C), row(R), segment(S).
path(h,C,R,C1,R,N) :- path(h,C+1,R,C1,R,N), adj(C,R,C+1,R),
   white(C,R), inroom(C,R,A), inroom(C+1,R,A),
   col(C;C1), row(R), num(N),N<3.
path(h,C,R,C1,R,N+1) :- path(h,C+1,R,C1,R,N), adj(C,R,C+1,R),
   white(C,R), inroom(C,R,A), inroom(C+1,R,B),
   A!=B, col(C;C1), row(R), num(N), N<3.
path(v,C,R,C,R1,N) :- path(v,C,R+1,C,R1,N), adj(C,R,C,R+1),
   white(C,R), inroom(C,R,A), inroom(C,R+1,A),
   col(C), row(R;R1), num(N),N<3.
path(v,C,R,C,R1,N+1) :- path(v,C,R+1,C,R1,N), adj(C,R,C,R+1),
   white(C,R), inroom(C,R,A), inroom(C,R+1,B),
   A!=B, col(C), row(R;R1), num(N), N<3.

% inroom(X,Y,A): grid cell (X,) is contained in Room A
inroom(X,Y,A) :- room(A,C,R,C1,R1), X<=C1, C<=X, Y<=R1, R<=Y,
   col(C;C1;X), row(Y;R1;R).

hide. show black(_,_).
```

Fig. 8. Representing Heyawake in the language of LPARSE: Part 2.

5 Bag Puzzle

Bag Puzzle (also called Corral Puzzle) is played on a rectangular grid. Initially some grid cells are numbered, like in Fig. 13. The goal is to draw a single, continuous loop along the lines of the grid, which contains all the numbers on the grid, like in Fig. 13, according to the rules below:

B1 The line should form a single loop.
B2 All numbered cells should be contained in the loop.
B3 If there is a numbered cell in the loop, the number of visible cells from that cell in all orthogonal directions should be the same as the number contained in it.

Deciding whether there is a solution to Bag Puzzle is NP-complete [3].

Main Program To compute a solution like in Fig. 13, in addition to the specific puzzle description, we need to describe Bag in the language of LPARSE, e.g., as in Fig.s 15 and 16. The first part of this formulation, about generating a single loop, is very similar to that of Masyu. On the other hand, the second part is different, since it involves the conditions about visibility, B2 and B3. In the second part, two auxiliary definitions

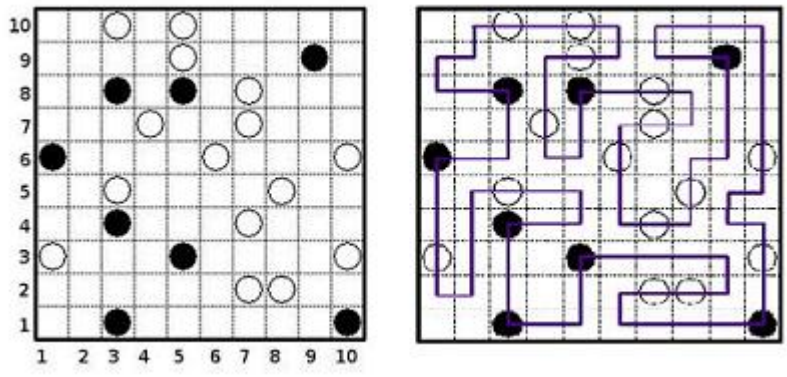

Fig. 9. A Masyu puzzle and its solution.

```
white(1,3). black(1,6). black(3,1). black(3,4). white(3,5).
black(3,8). white(3,10). white(4,7). black(5,3). black(5,8).
white(5,9). white(5,10). white(6,6). white(7,2). white(7,4).
white(7,7). white(7,8). white(8,2). white(8,5). black(9,9).
black(10,1). white(10,3). white(10,6).
```

Fig. 10. The input file describing the Masyu puzzle in Fig. 9.

are introduced: `visible` and `out`. The former describes the visibility of a cell from another one, and the latter checks whether a cell is outside the loop or not.

We describe Bag Puzzle in the language of LPARSE as follows.

Input and Output Suppose that we are given a Bag puzzle with a rectangular grid of size $n \times m$, containing some numbered cells. Each grid cell is identified by its lower left grid point; so a cell `(X,Y)` containing a number `N` is described by an atom of the form `cell(N,X,Y)`. For instance, the input file describing the Bag puzzle in Fig. 13 is presented in Fig. 14. Solutions to Bag puzzles can be characterized by atoms of the form `line(S,X,Y)` ("if `S=h` (resp. `S=v` then the horizontal (resp. vertical) segment `((X,Y),(X+1,Y))` (resp. `((X,Y),(X,Y+1))`) is contained in the drawn line").

6 Discussion

With the programs discussed above, we have experimented with some instances of the grid puzzles (Nurikabe, Heyawake, Masyu, Bag Puzzle) using SMODELS (Version 2.32), CLASP (Version 1.0.2), and CMODELS (Version 3.55) with ZCHAFF (Version 2007.3.12). All problem instances are available at `http://people.sabanciuniv.edu/~esraerdem/ASP-benchmarks/puzzles.html`. The experimental results are summarized in Tables 1 and 2. All CPU times are in seconds, for a workstation with a 1.5GHz Xeon processor and 4x512MB RAM, running Red Hat Enterprise Linux (Version 4.3). The computation times include the time spent for grounding; only the computation times less than 400 seconds are shown.

```
% File: masyu.lp

col(1..n). row(1..m). direction(v;h).

ball(X,Y) :- black(X,Y), col(X), row(Y).
ball(X,Y) :- white(X,Y), col(X), row(Y).

% Generate a set of unit segments passing through cells
{line(S,X,Y)} :- direction(S), col(X), row(Y).
:- line(h,n,Y), row(Y).
:- line(v,X,m), col(X).

% including balls:
:- ball(X,Y), not on(X,Y).

on(X,Y) :- line(S,X,Y), direction(S), col(X), row(Y).
on(X+1,Y) :- line(h,X,Y), col(X), row(Y), X<n.
on(X,Y+1) :- line(v,X,Y), col(X), row(Y), Y<m.

% ensuring the constraints M1--M3 are satisfied:

% M1:
% Every cell that the generated line passes through is
% connected to exactly 2 such grid cells:
:- 3 {line(h,X-1,Y), line(h,X,Y), line(v,X,Y), line(v,X,Y-1)},
   on(X,Y), col(X), row(Y).
:- {line(h,X-1,Y), line(h,X,Y), line(v,X,Y), line(v,X,Y-1)} 1,
   on(X,Y), col(X), row(Y).

% Furthermore, every cell on the generated line is reachable
% from other grid cells on the line:
:- on(X,Y), on(Z,W), not reachable(X,Y,Z,W),
   col(X;Z), row(Y;W).

adj(X,Y,X,Y+1) :- line(v,X,Y), col(X), row(Y), Y<m.
adj(X,Y,X+1,Y) :- line(h,X,Y), col(X), row(Y), X<n.
adj(X,Y,Z,W) :- adj(Z,W,X,Y), col(X;Z), row(Y;W).

reachable(X,Y,X,Y) :- on(X,Y), col(X;Z), row(Y;W).
reachable(X,Y,Z,W) :- reachable(I,J,Z,W), adj(X,Y,I,J),
   col(X;I;Z), row(Y;J;W).
```

Fig. 11. Representing Masyu in the language of LPARSE: Part 1.

Table 1. Experimental results: Problem size

Puzzle	Instance	Grid size	SMODELS		CMODELS			CLASP	
			# of atoms	# of rules	# of atoms	# of clauses	# of loop formulas	# of loop formulas	average loop size
Nurikabe	n1	10×10	20661	84410	63319	191567		35417	16.7
	n2		20667	84668	74642	235436		50736	17.1
	n3		20667	84671	62806	192199	8210	16450	12.4
	n4		20667	84669	75063	236212		26564	15.3
	n5		20673	84930	68775	208882	6320	11573	12.8
	n6		20664	84539	78402	247622		19052	14.2
	n7		20670	84803	66592	203623	4744	15793	13.3
	n8	15×15	102657	436972	438712	1414579		388361	20.7
	n9		102710	437248	426668	1365831			
	n10		102660	437808	380954	1211337		861656	23.5
Heyawake	h1	10×10	36213	70559	67976	176813	43	1827	33.9
	h2		36201	70572	70630	182483	44	2252	34
	h3		36205	70586	72953	192925	78	3057	28.2
	h4		36221	70816	71634	188768	49	1470	28.3
	h5		36221	70559	71466	188362	36	1626	27.2
	h6		36177	70573	72952	192889	60	5721	28.6
	h7		36139	70571	70794	186306	53	4309	31.7
	h8	15×15	172511	355712	359478	969960	100	7334	22.8
	h9		172545	355723	354414	955970	64	13643	22.6
	h10		172558	355717	362030	980198	75	12792	23.1
Masyu	m1	10×10	20696	1031458	56538	179956	145	3194	15.3
	m2		20710	1031496	57384	183801	221	911	12.6
	m3		20685	1031441	57363	185080	3708	3935	16.5
	m4		20694	1031476	57348	183552	1991	4056	14.6
	m5		20715	1031505	57385	184046	702	3174	12.6
	m6		20705	1031469	56131	176879	2600	2841	23.4
	m7		20700	1031473	57374	184579	759	2394	11.4
	m8	15×15	102759	11545746	293447	951564		141683	31.6
	m9		102770	11545763	292518	941199			
	m10		102783	11545804	292523	940514			
Bag	b1	10×10	59385	1833864	117498	321344	2527	16272	18.4

Table 2. Experimental results: Computation time

Puzzle	Instance	Grid size	CPU time (sec.)		
			SMODELS	CMODELS	CLASP
Nurikabe	n1	10×10	12		26
	n2		11		26
	n3		10		6
	n4		6		8
	n5		5		5
	n6		8		8
	n7		5		7
	n8	15×15			288
	n9				
	n10				363
Heyawake	h1	10×10	6	22	2
	h2		6	19	2
	h3		8	28	1
	h4		8	14	1
	h5		2	11	1
	h6		6	18	1
	h7		3	15	1
	h8	15×15	312	168	6
	h9		267	134	9
	h10		248	141	8
Masyu	m1	10×10	10	58	9
	m2		7	87	7
	m3		16		8
	m4		15		7
	m5		7	206	8
	m6		7		7
	m7		8	224	7
	m8	15×15			174
	m9		187		
	m10		198		
Bag	b1	10×10			24

```
% M3:
% The loop must turn (90 degrees) at black circles
:- black(X,Y), line(h,X,Y), line(h,X-1,Y), col(X), row(Y).
:- black(X,Y), line(v,X,Y), line(v,X,Y-1), col(X), row(Y).

% but it must travel straight through the next and previous
% cells in its path.
:- 1{line(v,X+1,Y), line(v,X+1,Y-1)},
   black(X,Y), line(h,X,Y), col(X), row(Y).
:- 1{line(v,X-1,Y), line(v,X-1,Y-1)},
   black(X,Y), line(h,X-1,Y), col(X),row(Y).

:- 1{line(h,X,Y+1), line(h,X-1,Y+1)},
   black(X,Y), line(v,X,Y), col(X), row(Y).
:- 1{line(h,X,Y-1), line(h,X-1,Y-1)},
   black(X,Y), line(v,X,Y-1), col(X), row(Y).

% M2:
% White circles must be traveled straight through
:- 1{line(v,X,Y),line(v,X,Y-1)}, 1{line(h,X,Y), line(h,X-1,Y)},
   white(X,Y), col(X), row(Y).

% but the loop must turn (90 degrees) in the previous
% and/or next cell in its path.
:- white(X,Y), line(h,X,Y), line(h,X+1,Y), line(h,X-1,Y),
   line(h,X-2,Y), col(X), row(Y).
:- white(X,Y), line(v,X,Y), line(v,X,Y+1), line(v,X,Y-1),
   line(v,X,Y-2), col(X), row(Y).

hide. show line(_,_,_).
```

Fig. 12. Representing Masyu in the language of LPARSE: Part 2.

In Table 1, the number of atoms in the propositional theory obtained by CMODELS includes the new atoms introduced for clausification. The number of loop formulas added by CMODELS is less than the one added by CLASP, but keep in mind that what is counted as loop formulas in CMODELS is different from that of CLASP. In CMODELS, it describes the number of "reason clauses" [6]; in CLASP, it describes the number of "loop nogoods" [5]. Compared to the other three puzzles, Nurikabe puzzles are more difficult for CMODELS since they require too many loop formulas.

Observe that most puzzles require too many loop formulas of large sizes (e.g., to solve n10 CLASP adds 861656 loop formulas of average size 24), unlike other nontight programs used as benchmarks (e.g., nontight Sokoban problems), which typically require couple of hundred loop formulas. In this sense, these puzzles will serve as a useful suite of benchmarks to evaluate algorithms introduced for handling nontight programs.

Observe also that representing these puzzles is indeed challenging (consider, e.g., conditions M1, N4, H5, B2, B3). Fortunately, the expressivity of ASP (e.g., being able to represent transitive closure, and cardinality constraints) makes it a bit easier.

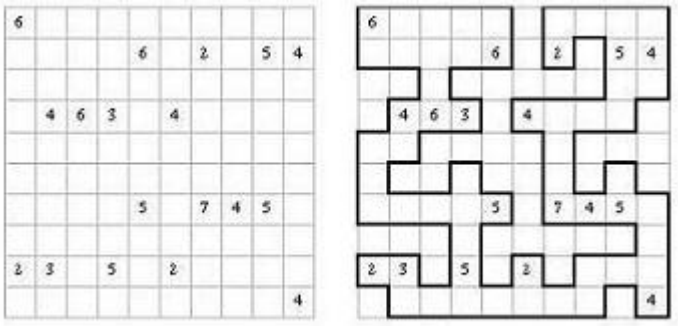

Fig. 13. A sample Bag Puzzle with its solution.

```
cell(4,0,9). cell(2,1,0). cell(3,1,1). cell(5,1,3). cell(4,8,9).
cell(2,1,5). cell(5,3,4). cell(7,3,6). cell(4,3,7). cell(6,9,0).
cell(5,3,8). cell(4,6,1). cell(6,6,2). cell(3,6,3). cell(4,6,5).
cell(6,8,4). cell(2,8,6). cell(5,8,8).
```

Fig. 14. The input file describing the Bag puzzle in Fig. 13.

Similarities between representations of the puzzles (e.g., generation of line segments, connectedness of nodes) suggest some extensions of the language of LPARSE (e.g., a template for transitive closure, like in the language of of DLT [1]). They also hint at two interesting problems: building a general grid puzzle solver (in the sense of the general tabular puzzle solver Constraint Lingo [2]), and the automatic generation of puzzle instances of various difficulty, using ASP methods. Both problems, in addition to formulating the puzzles in the language of DLV and DLT, are left as a future work.

Acknowledgments Thanks to Yuliya Lierler and Benjamin Kaufmann for answering our questions about the statistics printed out by CMODELS and CLASP.

References

1. F. Calimeri, G. Ianni, G. Ielpa, A. Pietramala, and M. Santoro. A system with template answer set programs. In *Proc. of JELIA*, 2004.
2. R.A. Finkel, V.W. Marek, and M. Truszczynski. Constraint lingo: Towards high-level constraint programming. *Software: Practice and Experience*, 34(15):1481–1504, 2004.
3. E. Friedman. Corral puzzles are NP-complete. 2002.
4. E. Friedman. Pearl puzzles are NP-complete. 2002.
5. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-Driven Answer Set Solving. In *Proc. of IJCAI*, 2007.
6. Y. Lierler. SAT-Based Answer Set Programming. 2007.
7. V. Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*. 1999.
8. V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. 1999.
9. B. McPhail. The complexity of puzzles: NP-completeness results for Nurikabe and Minesweeper. 2003.
10. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *AMAI*, 25:241–273, 1999.

```
% File: bag.lp

x(0..n). y(0..m). direction(v;h).

% Generate a set of unit segments along the grid lines
{line(S,X,Y)} :- direction(S), x(X), y(Y).
:- line(h,n,Y), y(Y).
:- line(v,X,m), x(X).

% ensuring the constraints B1--B3 are satisfied:

% B1:
% Every grid point on the generated line unit is connected to
% exactly 2 such grid points:
:- 3 {line(h,X-1,Y), line(h,X,Y), line(v,X,Y), line(v,X,Y-1)},
   on(X,Y), x(X), y(Y).
:- {line(h,X-1,Y), line(h,X,Y), line(v,X,Y), line(v,X,Y-1)} 1,
   on(X,Y), x(X), y(Y).

on(X,Y) :- line(S,X,Y), direction(S), x(X), y(Y).
on(X+1,Y) :- line(h,X,Y), x(X), y(Y), X<n.
on(X,Y+1) :- line(v,X,Y), x(X), y(Y), Y<m.

% Furthermore, every grid point on the generated line is
% reachable from other grid points on the line:
:- on(X,Y), on(Z,W), not reachable(X,Y,Z,W), x(X;Z), y(Y;W).

adj(X,Y,X,Y+1) :- line(v,X,Y), x(X), y(Y), Y<m.
adj(X,Y,X+1,Y) :- line(h,X,Y), x(X), y(Y), X<n.
adj(X,Y,Z,W) :- adj(Z,W,X,Y), x(X;Z), y(Y;W).

reachable(X,Y,X,Y) :- on(X,Y), x(X;Z),y(Y;W).
reachable(X,Y,Z,W) :- reachable(I,J,Z,W), adj(X,Y,I,J),
   x(X;I;Z), y(Y;J;W).
```

Fig. 15. Representing Bag in the language of LPARSE: Part 1.

```
% B3: If there is a numbered cell in the loop, the number of
%     visible cells from that cell in all orthogonal directions
%     should be the same as the number contained in it.
:- N {visible(X,Y,Z,T):x(Z):y(T)}, cell(N,X,Y), x(X), y(Y).
:- {visible(X,Y,Z,T):x(Z):y(T)} N-2, cell(N,X,Y), x(X), y(Y).

% visible0(X,Y,X1,Y1): cell (X,Y) is visible from
%                      adjacent cell (X1,Y1)
visible0(X,Y,X,Y+1) :- x(X), y(Y), not line(h,X,Y+1), Y<m-1.
visible0(X,Y,X,Y-1) :- x(X), y(Y), not line(h,X,Y), Y>0.
visible0(X,Y,X+1,Y) :- x(X), y(Y), not line(v,X+1,Y), X<n-1.
visible0(X,Y,X-1,Y) :- x(X), y(Y), not line(v,X,Y), X>0.

% visible(X,Y,X1,Y1): cell (X,Y) is visible from cell (X1,Y1)
visible(X,Y,Z,T) :- visible0(X,Y,Z,T),x(X;Z), y(Y;T).
visible(X,Y,Z,Y) :- visible(X,Y,Z+1,Y), visible0(Z,Y,Z+1,Y),
   x(X;Z), y(Y), Z<X-1, X>1, X<n, Y<m.
visible(X,Y,Z,Y) :- visible(X+1,Y,Z,Y), visible0 (X,Y,X+1,Y),
   x(X;Z), y(Y), Z>X+1, X<n-2, Y<m.
visible(X,Y,X,T) :- visible(X,Y+1,X,T), visible0(X,Y,X,Y+1),
   x(X), y(Y;T), T>Y+1, Y<m-2, X<n.
visible(X,Y,X,T) :- visible(X,Y,X,T+1), visible0(X,T,X,T+1),
   x(X), y(Y;T), T<Y-1, Y>1, Y<m, X<n.

% B2: No numbered cell is outside the loop.
:- cell(M,X,Y), out(X,Y).

out(X,Y) :- Y=m-1, X<n, cell(M,X,Y), not line(h,X,Y+1).
out(X,Y) :- Y=0, X<n, cell(M,X,Y), not line(h,X,Y).
out(X,Y) :- X=n-1, Y<m, cell(M,X,Y), not line(v,X+1,Y).
out(X,Y) :- X=0, Y<m, cell(M,X,Y), not line(v,X,Y).

out(X,Y) :- X<n-1, X>0, Y<m-1, Y>0, cell(M,X,Y),
   visible(X,Y,0,Y), not line(v,0,Y).
out(X,Y) :- X<n-1, X>0, Y<m-1, Y>0, Z=n-1, cell(M,X,Y),
   visible(X,Y,Z,Y), not line(v,n,Y).
out(X,Y) :- X<n-1, X>0, Y<m-1, Y>0, cell(M,X,Y),
   visible(X,Y,X,0), not line(h,X,0).
out(X,Y) :- X<n-1, X>0, Y<m-1, Y>0, T=m-1, cell(M,X,Y),
   visible(X,Y,X,T), not line(h,X,m).

hide. show line(_,_,_).
```

Fig. 16. Representing Bag in the language of LPARSE: Part 2.