ASP Encodings of Spinpossible

Valentin Mayer-Eichberger

IVU Traffice Technologies
Bundesallee 88, 12000 Berlin
vme@ivu.de

1 ASP encoding

1.1 Motivation for vector representation

The two following problems are essentially the same. Replacing -8 with 1 in the first and 3 and -4 in the second requires the same kind of spins.

-8	2	3		1	2	**4**
4	5	6		**-3**	5	6
7	**1**	9		7	8	9

Human players detect that instantly. We will explain in following two paragraphs how to arrive at a representation such that these two board and the necessary spins to solve them, are the same.

Vector representation: First we choose a different representation of the numbers on the boards. Each number is replaced by the vector pointing to where it should go, and the parity bit (if the number is upside down its 1, otherwise 0)

-8	2	3	(2,1,1)	(0,0,0)	(0,0,0)
4	5	6	(0,0,0)	(0,0,0)	(0,0,0)
7	**1**	9	(0,0,0)	(-1,-2,0)	(0,0,0)

E.g. the number in the upper left has to go down by two and one to the right.

Order on states: To further unify the two initial boards we need to define an ordering such that we rotate and reflect the rectangle to a canonical representation. Lets view the board as a vector of n numbers, e.g. the first board would be $(-8, 2, 3, 4, 5, 6, 7, 1, 9)$, or in the vector representation

$$((2,1,1),(0,0,0),(0,0,0),(0,0,0),(0,0,0),(0,0,0),(0,0,0),(-1,-2,0),(0,0,0))$$

Let the ordering $<_{lex}$ be the lexicographic ordering on the vector representation.

For a board there are 8 symmetric representation, generated by id,90, reflection, and combinations of these.

To find the canonical representation for a board is then to find the maximal element under the lexicographic ordering $<_{lex}$ among all possible symmetric translations of the board.

It remains to be shown that the overhead of estimating the correct turn and changing the board justifies the decrease in visited nodes. The basic idea is that the computed nogoods have a wider usage through out the search tree.

Fixing boarder rows/columns: The two boards from the motivation are now quite similar, but still not the same. To find the ultimate canonical representation we need to realize that the side column what is not used by the spins to solve such a board is not needed and can be omitted. This still has to be proven ...

If we remove the unused columns and rows before estimating the maximal element of the symmetric translations the boards should finally be the same. This also means that throughout the search the board decreases in size as we fix more and more rows and columns.

Decrease in state variables: Let the size of the board be $m \cdot n$. In this paragraph we will omit the move k. With a direct approach, to encode $m * n$ numbers with parity in $m * n$ positions requires $2 * m^2 \cdot n^2$ variables.

```
state(X,Y,V) :- m(X), n(Y), mn(V). % n*m*n*m*2
```

By using the vector representation we can break down the board into each dimension (x-axis, y-axis, parity), e.g.:

-8	2	3		2	0	0		1	0	0		1	0	0
4	5	6		0	0	0		0	0	0		0	0	0
7	**1**	9		0	-2	0		0	-1	0		0	0	0

representing these three boards as state variables we need the following predicates

```
state(x,X,Y,V) :- m(X), n(Y), m(V). %n*m*m
state(y,X,Y,V) :- m(X), n(Y), n(V). %n*m*n
parity(X,Y) :- m(X), n(Y).  %n*m
```

These are in total $n * m * (n + m)$ variables!

1.2 Motivation for recursive state update

What do the following problems have in common?

1	-8	3		-9	-8	-7
4	-5	6		-6	-5	-4
7	-2	9		-3	-2	-1

Both can be solved in one step. With spin $(1, 0)$ to $(1, 2)$ for the first and spin $(0, 0)$ to $(2, 2)$ for the second. But they have more in common: the middle column switches the same cells. This means the switches of the first spin are implied by the second spin. How do we use that in our encoding?

Using the breakdown of dimensions we encode a move just in one dimension (omitting step number and dimension):

```
switch(A+1,B-1) :- switch(A,B),A < B.   % (n^2+n)/2
```

Thus a state update is then

1.3 State update

State update in dimension x is as follows:

```
state(K+1,x,X2,Y2,X1-X2+V) :-  % k*m^3*n^4
       state(K,x,X1,Y1,V),
       switch(K+1,x,X1,X2),
       switch(K+1,y,Y1,Y2).
```