Tom Lang

CSE494 Final SUDOKU report

Background

Sudoku is a logic based number placement puzzle. There are many different types of puzzles. The most popular kind is a 9x9 square and the objective to solve this puzzle is to place the numbers 1-9 in each column and row and a 3x3 square and there can not be any repeats of a number in the same column, row or 3x3 sub square. This modern puzzle was invented by Howard Garns in 1979. Sudoku means single number. (Wikipedia)

There are many different types of suduko puzzles. Sudoku puzzles can range in overall size from a 9x9, 4x4, 16x16. With these puzzles each having a sub squares size of 3x3, 2x2, 8x8 respectively. Also with in suduko puzzles there can be sub squares that are not square but are just subgroups that are in random shapes (like a jig saw puzzle). There are also some puzzles that have small sub groups of 2-4 numbers that must add up to a certain value.

Since the original Sudoku is a well known and there are already many ways to solve this problem using answer set programming, I am going to take some variations of Sudoku and find solutions using answer set programming.

Problem description

I am going to incorporate 3 different types of Sudoku puzzles into one solution. I am going to take the traditional Sudoku puzzle with a size of 5x5. I am going to then have 5 different 5x5 puzzles with the 5th puzzle over lapping the other 4 puzzles. Figure 1 below shows the layout of the five Sudoku puzzles.

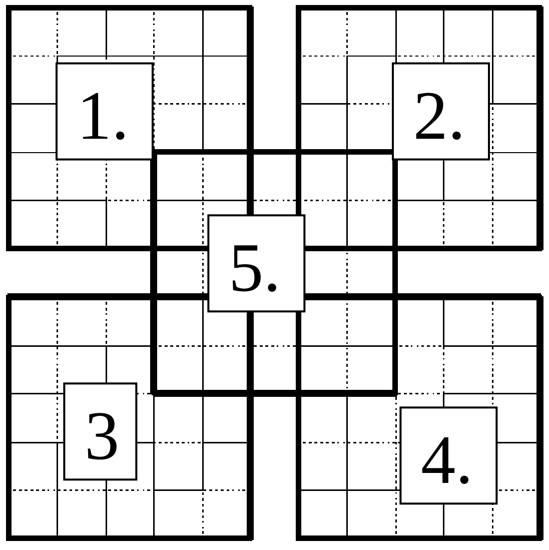

Figure 1

Figure 1 shows the layout of these 5 puzzles. The goal behind this is to place numbers 1-5 in each column and row. Also I have to find a solution so that the 2x2 squares that are overlapping puzzle 5 and the other 4 puzzles are the same.

To add to the complexity of this puzzle I am going to add another twist to this puzzle. Within each puzzle there will be sub groups of 2-4 elements and each sub group will have a value associated with it. The goal behind this is to place numbers in the sub group so that the values add up the value listed. Figure 2 below shows the puzzle layout with subgroups and the sums of the groups.

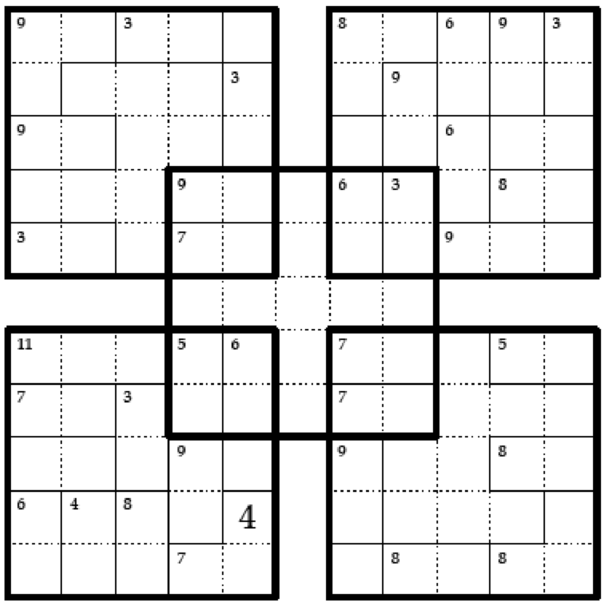
Figure 2

Figure 2 is what the end user would solve. Now I am going to use answer set programming and solve this puzzle. When I have completed this program anyone will be able to solve any puzzle that looks like figure 2.

Main Results

In this section of the report I will go through the code and give an explanation for each block of code. I will explain what I did and how it works. In this first section of code I have the user input and explain how the sub-groups should be set up so the program can find the correct solutions and also set up some variable ranges that will be used later on in the code. For the purpose of this section I am only putting small sections of code in; the code can be seen in its entirety at the end of this report in the appendix.

```
%This sets up variables to be used later in choice formulas and
eliminating solutions
number (1..5).
```

```
first(1..2).
last(4..5).
value(1..15).
%Where the user enters the given puzzle.
%They will enter any numbers already given in the puzzle
%They will then enter the sub groups and there sums
%The format for the sub groups is the X and Y for all cells in
the sub-group followed by the value if the sum
%Example sX(X1,Y1,X2,Y2....XN,YN,SUM)
c(5,2,4).
sa(1,4,1,5,2,5,9).
sa(3,5,4,5,3).
sa(5,3,5,4,3).
………
……….
se(5,4,5,5,3).
se(1,1,1,2,5).
se(2,1,2,2,6).
se(4,1,5,1,7).
se(4,2,5,2,7).
```

This was the best way I could find to have a user enter the input. There will be an explanation of the format of the sub-group input later in this section. Figure 3 below shows a small piece of puzzle C. This is the lower 2 rows of the puzzle with the X=1, Y=1 in the lower left hand corner.

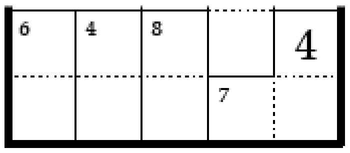

Figure 3

The example of how the user would input these two rows would be as follow. The first sub-group would be at x=1, y=1 and x=1, y=2.

```
sc(1,1,1,2,6).
```

The other 3 sub-groups look like this

```
sc(2,1,2,2,8).
sc(4,1,5,1,5,2,7).
```

The way you tell which 5x5 puzzle you are adding a sub-group to is by what letter you place after the s. So you can have sa, sb, sc, sd, se. These are your five options for adding sub groups to the puzzle. Also if you are given a number in the beginning like in figure 3, you would say what puzzle it is and follow it up with the x,y and the value you want to set at that location. The user input for the number 4 would be

```
c(5,2,4).
```

In this next section of code I set up a few domains to be used in the eliminating of solutions and also use the choice formulas to create all possible solutions for all five puzzles.

```
#domain
number(X;Y;N;X1;Y1;N1;X2;Y2;N2;X3;Y3;N3;X4;Y4;N4;X5;Y5;N5).
#domain value(V).
```

```
%generate all possible answers for the 5 sudoku puzzles
% a Represents the puzzle in the upper left
1 { a(X,Y,K) : number(K) } 1.

% b Represents the puzzle in the upper right
1 { b(X,Y,K) : number(K) } 1.
```

I found this was the best way to create all the possible solutions because after generating them I can then control which section of the puzzle I work on and eliminate solutions. This next section from the code doesn't have any function it is just an explanation of how the puzzles are setup. This shows how all five puzzles and how they are lettered and then for each puzzle the X shows where location X=1, Y=1. This was important for the code because now anyone else who reads my code will be able to understand my layout of the puzzle.

```
%letter representation of the puzzles.
%  A  B
%    E
%  C  D
%NOTE: for this program I am saying that the spot X=1 Y=1 is in
the lower left of the square
%     1  2   3   4   5
%     ------------------
%     5|                 |
%     4|                |
%     3|                |
%     2|                |
%     1|X               |
%     ------------------
```

In this next section of code I go through and eliminate all solutions that have the same number in each row or column. This will eliminate a lot of solutions that are not needed. This was done for all five puzzles.

```
%eliminate all answers with the same number listed more then once
%in a column and row
%each row and column has 1-5
%for the upper left puzzle
:- a(X1,Y,N), a(X2,Y,N), X1<X2.
:- a(X,Y1,N), a(X,Y2,N), Y1<Y2.

%for the upper right puzzle
:- b(X1,Y,N), b(X2,Y,N), X1<X2.
:- b(X,Y1,N), b(X,Y2,N), Y1<Y2.
```

In this next section of code I go through and eliminate all solutions that don't have the same numbers in the corners that overlap with the 5th puzzle (puzzle E). For each of the four underlying puzzles I take the corner that is overlapping with puzzle E and eliminate any solution that does not have the same numbers both in that puzzle and E.

```
%This section of codes makes sure the overlapping e puzzle
matches up with the 4 other puzzles it touches.
%if in the lower right hand corner of A
%and in the upper left hand corner of E
:- a(A+3,B,N), e(A,B+3,N1), first(A), first(B), N != N1.
```

```
        %if in the lower left hand corner of B
        %and in the upper right hand corner of E
        :- b(A,B,N), e(A+3,B+3,N1), first(A), first(B), N != N1.

        %if in the upper right hand corner of C
        %and in the lower left hand corner of E
        :- c(A+3,B+3,N), e(A,B,N1), first(A), first(B), N != N1.

        %if in the upper left hand corner of D
        %and in the lower right hand corner of E
        :- d(A,B+3,N), e(A+3,B,N1), first(A), first(B), N != N1.
```

In this last section of code I set up the eliminations of solutions that do not add up to the proper sub-group. These are the values that the user entered at the beginning of the program. I tried another way to solve this part of the puzzle but it was not user friendly. The other way I was considering solving the sub-groups was to have the puzzle be one big 11X11 square and then eliminate all the spaces where there were no numbers. The space between the individual puzzles (see figure 2). This would have allowed the user not to have to worry about 5 different puzzles, instead they would just have to know where X1, Y1 was and then from there could enter in the sub-groups. The problem with this was that it go confusing when trying to enter the numbers and I actually entered the numbers in wrong the first time. So that is why I decided to go with the solution where the user enters the sub-groups for all 5 individual puzzles. This was a lot easier for user input. All they had to know was what letter each puzzle represented and then just followed the example I gave in the code.

```
%This section accounts for the sub groups and the sub group must add up
the total
%NOTE: i am assuming that the sub group size is <=5
%Example in puzzle a spots (1,5),(2,5),(1,4) must add up to 9
%sa(X,Y,V)
%X,Y coordinate in puzzle
%V: value the group must add to

%Puzzle a
%sub group has 1 number in it
:- sa(X,Y,V),a(X,Y,N),N != V.

%sub group has 2 number in it
:- sa(X1,Y1,X2,Y2,V),a(X1,Y1,N1), a(X2,Y2,N2), (N1+N2) != V.

%sub group has 3 numbers in it
:- sa(X1,Y1,X2,Y2,X3,Y3,V),a(X1,Y1,N1), a(X2,Y2,N2),a(X3,Y3,N3),
(N1+N2+N3) != V.

%sub group has 4 numbers in it
:- sa(X1,Y1,X2,Y2,X3,Y3,X4,Y4,V),a(X1,Y1,N1),
a(X2,Y2,N2),a(X3,Y3,N3),a(X4,Y4,N4), (N1+N2+N3+N4) != V.

%sub group has 5 numbers in it
:- sa(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,V),a(X1,Y1,N1),
a(X2,Y2,N2),a(X3,Y3,N3),a(X4,Y4,N4),a(X5,Y5,N4),(N1+N2+N3+N4+N5) != V.
```

This section of code was then done 5 more times, for each of the 4 other puzzles to add up and eliminate all solutions that do not have the proper values in the sub group.

Solution

When you run the code in a command window, this is the solution that is given.

```
C:\Documents and Settings\tlang1\My Documents\Fall08\CSE494>lparse.exe
project.txt | smodels.exe 0
smodels version 2.26. Reading...done
Answer: 1
Stable Model: e(1,1,2) e(1,2,3) e(1,3,1) e(1,4,4) e(1,5,5) e(2,1,5)
e(2,2,1) e(2
,3,2) e(2,4,3) e(2,5,4) e(3,1,1) e(3,2,4) e(3,3,5) e(3,4,2) e(3,5,3)
e(4,1,3) e(
4,2,2) e(4,3,4) e(4,4,5) e(4,5,1) e(5,1,4) e(5,2,5) e(5,3,3) e(5,4,1)
e(5,5,2) d
(1,1,1) d(1,2,5) d(1,3,4) d(1,4,3) d(1,5,2) d(2,1,3) d(2,2,1) d(2,3,2)
d(2,4,4)
d(2,5,5) d(3,1,5) d(3,2,4) d(3,3,1) d(3,4,2) d(3,5,3) d(4,1,2) d(4,2,3)
d(4,3,5)
 d(4,4,1) d(4,5,4) d(5,1,4) d(5,2,2) d(5,3,3) d(5,4,5) d(5,5,1)
c(5,2,4) c(1,1,4
) c(1,2,2) c(1,3,1) c(1,4,3) c(1,5,5) c(2,1,3) c(2,2,1) c(2,3,5)
c(2,4,4) c(2,5,
2) c(3,1,5) c(3,2,3) c(3,3,2) c(3,4,1) c(3,5,4) c(4,1,1) c(4,2,5)
c(4,3,4) c(4,4
,2) c(4,5,3) c(5,1,2) c(5,3,3) c(5,4,5) c(5,5,1) b(1,1,5) b(1,2,1)
b(1,3,4) b(1,
4,3) b(1,5,2) b(2,1,1) b(2,2,2) b(2,3,5) b(2,4,4) b(2,5,3) b(3,1,3)
b(3,2,4) b(3
,3,2) b(3,4,1) b(3,5,5) b(4,1,2) b(4,2,3) b(4,3,1) b(4,4,5) b(4,5,4)
b(5,1,4) b(
5,2,5) b(5,3,3) b(5,4,2) b(5,5,1) a(1,1,1) a(1,2,3) a(1,3,5) a(1,4,2)
a(1,5,4) a
(2,1,2) a(2,2,1) a(2,3,4) a(2,4,5) a(2,5,3) a(3,1,5) a(3,2,2) a(3,3,3)
a(3,4,4)
a(3,5,1) a(4,1,4) a(4,2,5) a(4,3,1) a(4,4,3) a(4,5,2) a(5,1,3) a(5,2,4)
a(5,3,2)
 a(5,4,1) a(5,5,5)
False
Duration 0.359
Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 942
Number of rules: 4940
Number of picked atoms: 565
Number of forced atoms: 141
Number of truth assignments: 10589
Size of searchspace (removed): 0 (0)
```

Milestones

These are the milestones for this project. The first milestone was to sit down with the suduko puzzle and to solve the puzzle without any program assistance. It took me about 15 minutes to solve the puzzle by hand. It took me a couple minutes at first on exactly how to solve this puzzle. This was the first time I have ever solved a puzzle like this. The second milestone is to sit down and write pseudo code for how I wanted the code to be written.

The pseudo code that I wrote for this was first finding all possible solutions. Second to eliminate all solutions that have the same number in the same column and row. After I find these solutions I then find solutions that have the same overlap of the 5 puzzle and the other four puzzles. See figure 1 for exact layout. The last part of my pseudo code included taking all the of the sub groups and to eliminate all possibilities that do not add up to each given value. To do this I plan on taking all 5 puzzles separately and going through and adding up each sub group and if it doesn't match the given value I will eliminate it. The way I plan on adding up each sub group is to take in all the x and y locations and the last value is the sum. I will do this for sub groups that are broken up into 1 through 5.

After writing the pseudo code, I took my plan and started coding. The next milestone was when I created all possible solutions for the 5 sudoku puzzles. These solutions were only for a 5x5 square. I did not take into account any other requirements yet. The next milestone is when I get rid of all duplicate solutions. This includes getting rid of the solutions that have the same number twice in the same row or column. The next milestone after that was to take the 5th puzzle and over lap the corners of 5th puzzle with the 2x2 corners of the other four puzzles. I need to make sure the corners of the 5th puzzle have the same values in them as the respected corner of the it is over lapping. The second to last milestone of this program is to take the original puzzle and input all the sub groups and test the code and verify that the solution is the same one I calculated by hand. The final milestone of this project is to take the code and make sure it is optimized as much as possible.

Future work

After I get this program working as I have described there is some future work that could be done to optimize the project. This would include being able to change the size of the sudoku squares, for example a 6x6,7x7 or 8x8 square. Also being able to change the number of squares in the 5th puzzle, that over lap the corners of the 4 other puzzles. Also I would like to have a user face set up so the user can solve a puzzle that only has 1 of the variations of the puzzle I am solving, for example the original 5x5 sudoku, or the kakuro. The final future work that could be implemented on this code would be to have a graphical user interface. A GUI that would allow the user to click on a square and make it into sub-group or assign it a number. This would save the user from having to edit the lparse file. Also from the GUI the user could input the size of the puzzle and solve one the three individually or all three at once.

Self Assessment

After completing this project I went back and look at my project proposal and I am satisfied with what I accomplished. I completed everything I set out to do and I also learned a lot more about answer set programming. I learned that there are many different ways to code something and even though you might think one way is best. For example with my sub-groups, I thought I had a solid way of doing it with treating the puzzle as one big square. But it didn't work out and that is ok, I learned to try something else and it actually worked out a lot better. I was happy to see that when I took all three puzzles and put them together to solve one big puzzle everything worked and I didn't have to spend a lot of time debugging the code. I think was partly because I spent a little more time in the planning and pseudo code.

APPENDIX

References

wikipedia, "Sudoku." wikipedia. 12 NOV 2008. 12 Nov 2008 <http://en.wikipedia.org/wiki/Sudoku>.

CODE

```
%This sets up variables to be used later in choice formulas and eliminating solutions
number(1..5).
first(1..2).
last(4..5).
value(1..15).

%Where the user enters the given puzzle.
%They will enter any numbers already given in the puzzle
%They will then enter the sub groups and there sums
%The format for the sub groups is the X and Y for all cells in the sub-group followed by the value if the
sum
%Example sX(X1,Y1,X2,Y2....XN,YN,SUM)
c(5,2,4).
sa(1,4,1,5,2,5,9).
sa(3,5,4,5,3).
sa(5,3,5,4,3).
sa(1,3,2,3,9).
sa(1,1,2,1,3).
sa(4,1,5,1,7).
sa(4,2,5,2,9).
sb(1,4,1,5,2,5,8).
sb(3,4,3,5,6).
sb(4,4,4,5,9).
sb(5,4,5,5,3).
sb(2,3,2,4,9).
sb(3,2,3,3,6).
sb(1,1,1,2,6).
sb(2,1,2,2,3).
sb(4,2,5,2,8).
sb(1,3,1,4,1,5,9).
sc(1,5,2,5,3,5,11).
sc(4,4,4,5,5).
sc(5,4,5,5,6).
sc(1,4,2,4,7).
sc(3,3,3,4,3).
sc(4,2,4,3,9).
sc(1,1,1,2,6).
sc(2,1,2,2,4).
sc(3,1,3,2,8).
sc(4,1,5,1,5,2,7).
sd(1,5,2,5,7).
sd(1,4,2,4,7).
sd(4,5,5,5,5).
sd(1,2,1,3,9).
sd(4,3,5,3,8).
sd(2,1,3,1,8).
sd(4,1,5,1,5,2,8).
se(1,5,2,5,9).
se(1,4,2,4,7).
se(4,4,4,5,6).
se(5,4,5,5,3).
se(1,1,1,2,5).
se(2,1,2,2,6).
se(4,1,5,1,7).
se(4,2,5,2,7).


#domain number(X;Y;N;X1;Y1;N1;X2;Y2;N2;X3;Y3;N3;X4;Y4;N4;X5;Y5;N5).
#domain value(V).
%generate all possible answers for the 5 sudoku puzzles
% a Represents the puzzle in the upper left
1 { a(X,Y,K) : number(K) } 1.

% b Represents the puzzle in the upper right
1 { b(X,Y,K) : number(K) } 1.

% c Represents the puzzle in the lower left
1 { c(X,Y,K) : number(K) } 1.

% d Represents the puzzle in the lower right
1 { d(X,Y,K) : number(K) } 1.

% e Represents the puzzle in the middle
1 { e(X,Y,K) : number(K) } 1.

%letter representation  of the puzzles.
%  A      B
%    E
%  C      D
%NOTE: for this program i am saying that the spot X=1 Y=1 is in the lower left of the square
%         1  2   3   4   5
%         ------------------
%     5|                 |
%     4|                 |
%     3|                 |
%     2|                 |
%     1|X                |
%         ------------------

%eliminate all answers with the same number listed more then once in a column and row
%each row and column has 1-5
%for the upper left puzzle
:- a(X1,Y,N), a(X2,Y,N), X1<X2.
:- a(X,Y1,N), a(X,Y2,N), Y1<Y2.

%for the upper right puzzle
:- b(X1,Y,N), b(X2,Y,N), X1<X2.
:- b(X,Y1,N), b(X,Y2,N), Y1<Y2.

%for the lower left puzzle
:- c(X1,Y,N), c(X2,Y,N), X1<X2.
:- c(X,Y1,N), c(X,Y2,N), Y1<Y2.

%for the lower right puzzle
:- d(X1,Y,N), d(X2,Y,N), X1<X2.
:- d(X,Y1,N), d(X,Y2,N), Y1<Y2.

%for the puzzle in the middle
:- e(X1,Y,N), e(X2,Y,N), X1<X2.
:- e(X,Y1,N), e(X,Y2,N), Y1<Y2.
```

```
%This section of codes makes sure the overlapping e puzzle matches up with the 4 other puzzles it touches.
%if in the lower right hand corner of A
%and in the upper left hand corner of E
:- a(A+3,B,N), e(A,B+3,N1), first(A), first(B), N != N1.

%if in the lower left hand corner of B
%and in the upper right hand corner of E
:- b(A,B,N), e(A+3,B+3,N1), first(A), first(B), N != N1.

%if in the upper right hand corner of C
%and in the lower left hand corner of E
:- c(A+3,B+3,N), e(A,B,N1), first(A), first(B), N != N1.

%if in the upper left hand corner of D
%and in the lower right hand corner of E
:- d(A,B+3,N), e(A+3,B,N1), first(A), first(B), N != N1.


%This section accounts for the sub groups and the sub group must add up the total
%NOTE: i am assuming that the sub group size is <=5
%Example in puzzule a spots (1,5),(2,5),(1,4) must add up to 9
%sa(X,Y,V)
%X,Y coodinate in puzzle
%V:value the group must add to

%Puzzle a
%sub group has 1 number in it
:- sa(X,Y,V),a(X,Y,N),N != V.

%sub group has 2 number in it
:- sa(X1,Y1,X2,Y2,V),a(X1,Y1,N1), a(X2,Y2,N2), (N1+N2) != V.

%sub group has 3 numbers in it
:- sa(X1,Y1,X2,Y2,X3,Y3,V),a(X1,Y1,N1), a(X2,Y2,N2),a(X3,Y3,N3), (N1+N2+N3) != V.

%sub group has 4 numbers in it
:- sa(X1,Y1,X2,Y2,X3,Y3,X4,Y4,V),a(X1,Y1,N1), a(X2,Y2,N2),a(X3,Y3,N3),a(X4,Y4,N4),
(N1+N2+N3+N4) != V.

%sub group has 5 numbers in it
:- sa(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,V),a(X1,Y1,N1),
a(X2,Y2,N2),a(X3,Y3,N3),a(X4,Y4,N4),a(X5,Y5,N4),(N1+N2+N3+N4+N5) != V.

%Puzzle b
%sub group has 1 number in it
:- sb(X,Y,V),b(X,Y,N),N != V.

%sub group has 2 number in it
:- sb(X1,Y1,X2,Y2,V),b(X1,Y1,N1), b(X2,Y2,N2), (N1+N2) != V.

%sub group has 3 numbers in it
:- sb(X1,Y1,X2,Y2,X3,Y3,V),b(X1,Y1,N1), b(X2,Y2,N2),b(X3,Y3,N3), (N1+N2+N3) != V.

%sub group has 4 numbers in it
```

```
:- sb(X1,Y1,X2,Y2,X3,Y3,X4,Y4,V),b(X1,Y1,N1), b(X2,Y2,N2),b(X3,Y3,N3),b(X4,Y4,N4),
(N1+N2+N3+N4) != V.

%sub group has 5 numbers in it
:- sb(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,V),b(X1,Y1,N1),
b(X2,Y2,N2),b(X3,Y3,N3),b(X4,Y4,N4),b(X5,Y5,N4),(N1+N2+N3+N4+N5) != V.

%Puzzle c
%sub group has 1 number in it
:- sc(X,Y,V),c(X,Y,N),N != V.

%sub group has 2 number in it
:- sc(X1,Y1,X2,Y2,V),c(X1,Y1,N1), c(X2,Y2,N2), (N1+N2) != V.

%sub group has 3 numbers in it
:- sc(X1,Y1,X2,Y2,X3,Y3,V),c(X1,Y1,N1), c(X2,Y2,N2),c(X3,Y3,N3), (N1+N2+N3) != V.

%sub group has 4 numbers in it
:- sc(X1,Y1,X2,Y2,X3,Y3,X4,Y4,V),c(X1,Y1,N1), c(X2,Y2,N2),c(X3,Y3,N3),c(X4,Y4,N4),
(N1+N2+N3+N4) != V.

%sub group has 5 numbers in it
:- sc(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,V),c(X1,Y1,N1),
c(X2,Y2,N2),c(X3,Y3,N3),c(X4,Y4,N4),c(X5,Y5,N4),(N1+N2+N3+N4+N5) != V.


%Puzzle d
%sub group has 1 number in it
:- sd(X,Y,V),d(X,Y,N),N != V.

%sub group has 2 number in it
:- sd(X1,Y1,X2,Y2,V),d(X1,Y1,N1), d(X2,Y2,N2), (N1+N2) != V.

%sub group has 3 numbers in it
:- sd(X1,Y1,X2,Y2,X3,Y3,V),d(X1,Y1,N1), d(X2,Y2,N2),d(X3,Y3,N3), (N1+N2+N3) != V.

%sub group has 4 numbers in it
:- sd(X1,Y1,X2,Y2,X3,Y3,X4,Y4,V),d(X1,Y1,N1), d(X2,Y2,N2),d(X3,Y3,N3),d(X4,Y4,N4),
(N1+N2+N3+N4) != V.

%sub group has 5 numbers in it
:- sd(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,V),d(X1,Y1,N1),
d(X2,Y2,N2),d(X3,Y3,N3),d(X4,Y4,N4),d(X5,Y5,N4),(N1+N2+N3+N4+N5) != V.

%Puzzle e
%sub group has 1 number in it
:- se(X,Y,V),e(X,Y,N),N != V.

%sub group has 2 number in it
:- se(X1,Y1,X2,Y2,V),e(X1,Y1,N1), e(X2,Y2,N2), (N1+N2) != V.

%sub group has 3 numbers in it
:- se(X1,Y1,X2,Y2,X3,Y3,V),e(X1,Y1,N1), e(X2,Y2,N2),e(X3,Y3,N3), (N1+N2+N3) != V.

%sub group has 4 numbers in it
```

```
:- se(X1,Y1,X2,Y2,X3,Y3,X4,Y4,V),e(X1,Y1,N1), e(X2,Y2,N2),e(X3,Y3,N3),e(X4,Y4,N4),
(N1+N2+N3+N4) != V.

%sub group has 5 numbers in it
:- se(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,V),e(X1,Y1,N1),
e(X2,Y2,N2),e(X3,Y3,N3),e(X4,Y4,N4),e(X5,Y5,N4),(N1+N2+N3+N4+N5) != V.



hide.
show a(_,_,_).
show b(_,_,_).
show c(_,_,_).
show d(_,_,_).
show e(_,_,_).
```