Magic Square ASP Solver

CS449 Course Project
Project Guide :- Prof. G. Sivakumar

Swapnoneel Kayal - 200100154

Outline

1. Problem Statement
2. Work done so far
    a. Pre Midterm Task
    b. Post Midterm Tasks
    c. Output screenshots
3. Final Work
    a. Post Endterm Tasks
    b. Output screenshots

Problem Statement

ASP stands for Answer Set Programming. These solvers code the rules for a puzzle in the language of ASP, like clingo to solve them, These puzzles generally have a light python wrapper around them for easily reading the puzzles from file and outputting the solutions nicely. These solvers will either solve the puzzle completely, or give up and produce nothing.

In this project, we seek to make an ASP solver for the magic square problem initially and then build upon the level of difficulty by bringing in some additional requirements that the ASP solver needs to adhere to. These requirements will be presented in the upcoming slides.

Magic Square Problem :

A magic square of order n is an arrangement of n^2 distinct numbers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum up to the same constant known as the magic constant (M). A magic square contains the integers from 1 to n^2.

$$M = n*(n^2+1)/2$$

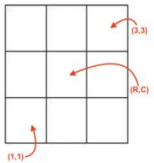
Work done so far

Pre-Midterm Task

Before the midterm review, we were able to accomplish the following :-

1. Lay down the grid setup for the puzzle which involves encoding the magic square in a structure that allows us to easily code the rules and subsequently apply constraints that suit the language of ASP

In case of magic square, I have indexed each cell as (R, C) where
$R \rightarrow$ row number $\in \{1, 2, ..n\}$
$C \rightarrow$ column number $\in \{1, 2, ..n\}$



2. Get hold of the puzzle definition in clingo

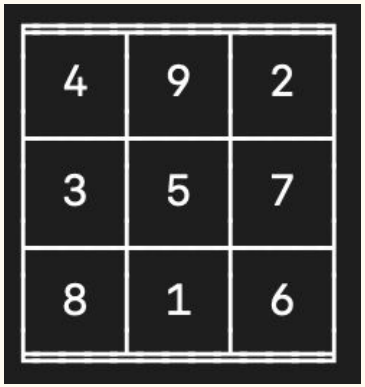
- Every cell of grid is filled with a number between 1 and n^2
- Different squares are filled with different numbers
- Every row sums up to the magic constant
- Every column sums up to the magic constant
- Both diagonals sum up to the magic constant

3. Python wrapper

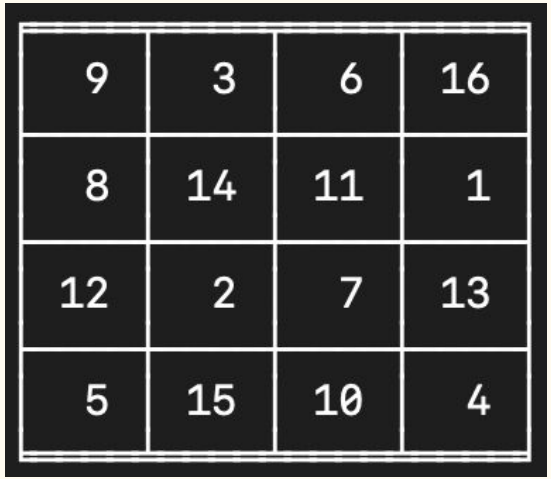
- This was achieved by leveraging the Python subprocess module
- To output the magic square in a readable manner, the tabulate library was used.

Output Screenshots

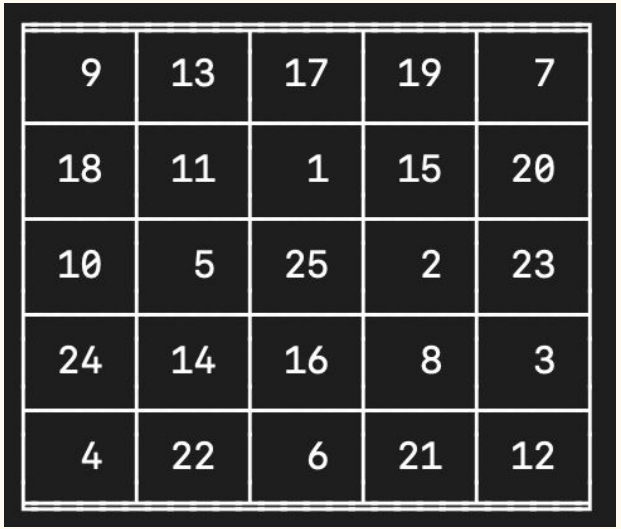
For n = 3:



For n = 4:



For n = 5:



This code is made available here :
https://github.com/codeWITHswap/CS449_TAIP/tree/main/Midterm-Review

Post-Midterm Tasks

During the midterm review, we were advised to work on incorporating the following additional functionalities in our ASP solver :

1. The user should be able to specify the cell at which they do not want a particular number
2. The user should be able to specify the cell at which they would like a particular number
3. The user should be able to define the magic constant for the square

We were able to tackle this in the following way:-

(1) Let us assume the user does not want the magic square of n = 3 to have the number 4 in the cell corresponding to the 1st row and 1st column. They can then specify this in the following format :

(row, column, number to be avoided)

This can be taken from the user in the form of text file during runtime. A sample constraints.txt in this case could look something like this :

(1,1,4)
(2,3,8)

Post-Midterm Tasks

(1) These numbers can then be processed into the following form by manipulating the strings :

:- num(1,1,4).

:- num(2,3,8).

These then get appended at the end of the standard magic.lp file and then a subprocess is invoked to execute the “modified” magic.lp file and the output is then taken by the wrapper.py to produce it in a readable manner using the tabulate library. Note that after the output is displayed to the user, the added lines in the “modified” magic.lp is removed and the original magic.lp is retrieved.

(2) In the previous case, we dealt with the user specifying the cell at which they do not want a number. Now, we will try look at it’s counterpart. What if the user wants a specific number at a particular cell?

Well, we can take the input in a similar manner as before i.e. in the following format :

(row, column, number to be kept)

This can be taken from the input in the form of a text file during runtime. A sample constraints.txt in this case could contain something like this :

(1,1,4)
(2,3,8)

Post Midterm Tasks

(2) These numbers can then be processed in into the following form by manipulating the strings :

num(1,1,4).

num(2,3,8).

These then get appended at the end of the standard magic.lp file and then a subprocess is invoked to execute the “modified” magic.lp file and the output is then taken by the wrapper.py to produce it in a readable manner using the tabulate library provided by python. Note that after the output is displayed to the user, the added lines in the “modified” magic.lp is removed and the original magic.lp is retained.

(3) The next task was to allow the user to mention the magic constant for the square. For this we had to change the internal structure of our magic.lp file. Previously, the cells got filled from range 1,2,..,n^2. But, now the cells get filled from the range 1,2,..N^2 where N is the smallest number from the set {1,2,..,10} that satisfies N*(N*N+1)/2 $>$ sum.

We had to define a finite set from which N has to be sampled because clingo does not like to work in boundless domains.

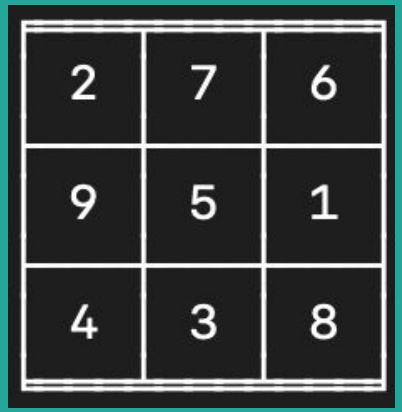
Note that the sum is the user defined magic constant. Appropriate changes were then made to make the sum of rows, columns and the two diagonals equal this sum.

Redefining the Tasks

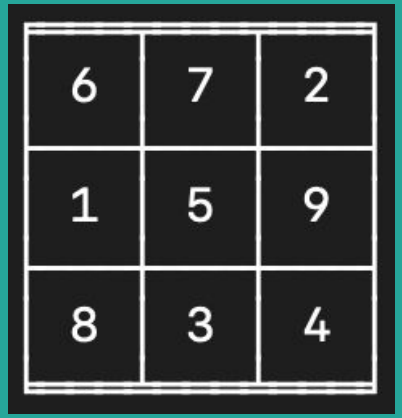
Task 1 - User can specify the number(s) they do not want at a particular cell(s) as well as the magic constant

Task 2 - User can specify the number(s) they want at a particular cell(s) as well as the magic constant

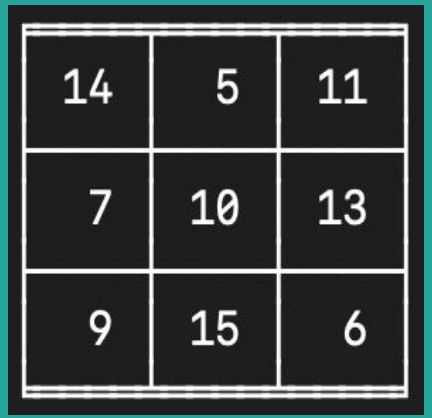
Output Screenshots for Task 1



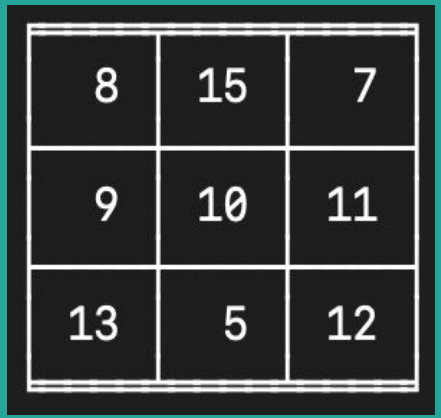
A 3x3 magic square without any constraints



A 3x3 magic square with (1,1,2)



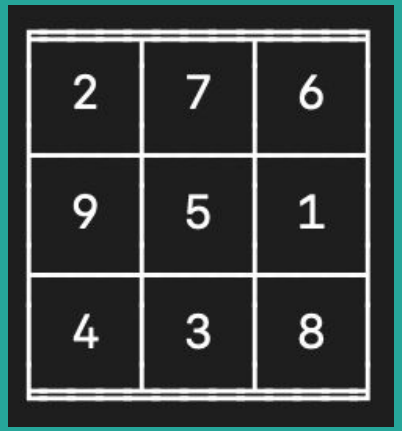
A 3x3 magic square with magic constant = 30



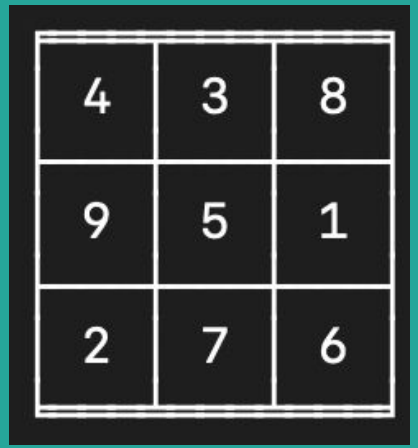
A 3x3 magic square with magic constant = 30 and (1,1,14)

This code is made available here :
https://github.com/codeWITHswap/CS449_TAIP/tree/main/Endterm-Review/Task-1

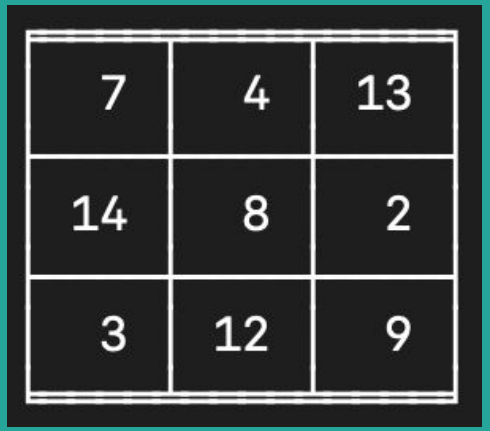
Output Screenshots for Task 2



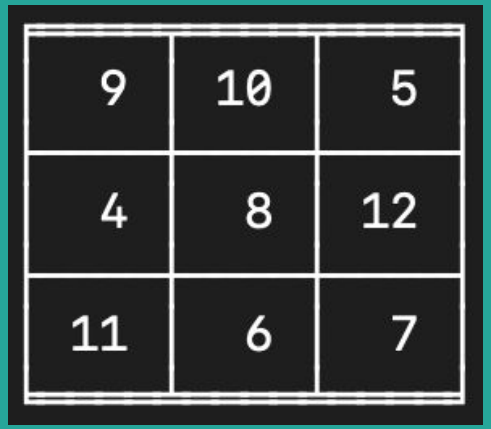
A 3x3 magic square without any constraints



A 3x3 magic square with (1,1,4)



A 3x3 magic square with magic constant = 24



A 3x3 magic square with magic constant = 24 and (1,1,9) and (2,3,12)

This code is made available here :
https://github.com/codeWITHswap/CS449_TAIP/tree/main/Endterm-Review/Task-2

Final Work - Post Endterm Tasks

Ramanujan's Magic Square

- 4x4 square
- Sum of numbers of any row is 139.
- Sum of numbers of any column is 139.
- Sum of numbers of any diagonal is 139.
- Sum of corner numbers is 139.
- Sum of the identical coloured boxes as shown on the right is also 139!

My output :-

22	12	18	87
74	31	27	7
26	8	73	32
17	88	21	13

Notice the numbers in the first row:
22 December 1887 was when, the great Indian Mathematician, Srinivasa Ramanujan was born!

22	12	18	87
88	17	9	25
10	24	89	16
19	86	23	11

22	12	18	87
88	17	9	25
10	24	89	16
19	86	23	11

22	12	18	87
88	17	9	25
10	24	89	16
19	86	23	11

22	12	18	87
88	17	9	25
10	24	89	16
19	86	23	11

22	12	18	87
88	17	9	25
10	24	89	16
19	86	23	11

```
1 {num(R, C, 1..12*12)} 1 :- R=1..4, C=1..4.

R1 = R2 :- num(R1,_,X), num(R2,_,X).
C1 = C2 :- num(_,C1,X), num(_,C2,X).

#const magic=139.

:- #sum{X : num(R,_,X)} != magic, R=1..4.
:- #sum{X : num(_,C,X)} != magic, C=1..4.

:- #sum{X : num(R,R,X)} != magic.
:- #sum{X : num(R,4+1-R,X)} != magic.

:- #sum{X : num(1,1,X;1,4,X;4,1,X;4,4,X)} != magic.
:- #sum{X : num(2,2,X;2,3,X;3,2,X;3,3,X)} != magic.

:- #sum{X : num(1,1,X;1,2,X;2,1,X;2,2,X)} != magic.
:- #sum{X : num(3,1,X;3,2,X;4,1,X;4,2,X)} != magic.

:- #sum{X : num(1,3,X;1,4,X;2,3,X;2,4,X)} != magic.
:- #sum{X : num(3,3,X;3,4,X;4,3,X;4,4,X)} != magic.

:- #sum{X : num(1,2,X;1,3,X;4,2,X;4,3,X)} != magic.
:- #sum{X : num(2,1,X;3,1,X;2,4,X;3,4,X)} != magic.

:- #sum{X : num(1,2,X;2,1,X;3,4,X;4,3,X)} != magic.
:- #sum{X : num(1,3,X;3,1,X;2,4,X;4,2,X)} != magic.

:- #sum{X : num(2,1,X;2,2,X;3,1,X;3,2,X)} != magic.
:- #sum{X : num(2,3,X;2,4,X;3,3,X;3,4,X)} != magic.
```

CODE

What does the following line mean?

:- #sum{X : num(1,1,X;1,4,X;4,1,X;4,4,X)} != magic.

This means that the final stable model can not include that combination wherein the sum of all the X's for which num(1,1,X) and num(1,4,X) and num(4,1,X) and num(4,4,X) are a part of the stable solution is not equal to 139

This ultimately ensures that the sum of the 4 corners of the magic square is 139

Maximum Even Magic Square

- The user is allowed to specify the value of ‘n’ i.e. #rows = #columns as well as the value of ‘sum’ i.e. the value of magic constant.
- However, we want to optimize (i.e. find the “best” solution among the set of “good” solutions) on the basis of maximum number of even numbers used.

CODE

```
1 {num(R, C, 1..12*12)}1 :- R=1..n, C=1..n.

R1 = R2 :- num(R1,_,X), num(R2,_,X).
C1 = C2 :- num(_,C1,X), num(_,C2,X).

#const magic=m.

:- #sum{X : num(R,_,X)} != magic, R=1..n.
:- #sum{X : num(_,C,X)} != magic, C=1..n.

:- #sum{X : num(R,R,X)} != magic.
:- #sum{X : num(R,n+1-R,X)} != magic.

#maximize{1,X : num(R,C,X), R=1..n, C=1..n, X\2=0}.
```

```
That stable model is chosen which has the maximum number of X's which satisfy
the following conditions->
(1) num(R,C,X) is a part of the stable model with R ∈ {1..n} and C ∈ {1..n}
(2) X when divided by 2 leaves a remainder of 0 implying X is an even number
```

OUTPUT

6	22	17
26	15	4
13	8	24

Maximum Prime Magic Square

- The user is allowed to specify the value of ‘n’ i.e. #rows = #columns as well as the value of ‘sum’ i.e. the value of magic constant.
- However, we want to optimize (i.e. find the “best” solution among the set of “good” solutions) on the basis of maximum number of prime numbers used.

CODE

```
1 {num(R, C, 1..12*12)} 1 :- R=1..n, C=1..n.

R1 = R2 :- num(R1,_,X), num(R2,_,X).
C1 = C2 :- num(_,C1,X), num(_,C2,X).

#const magic=m.

:- #sum{X : num(R,_,X)} != magic, R=1..n.
:- #sum{X : num(_,C,X)} != magic, C=1..n.

:- #sum{X : num(R,R,X)} != magic.
:- #sum{X : num(R,n+1-R,X)} != magic.

composite(N) :- N=1..12*12, I=2..N-1, N\I=0.
prime(N) :- N=2..n, not composite(N).
#maximize{1,X : num(R,C,X), R=1..n, C=1..n, prime(X)}.

#show num/3.
```

```
That stable model is chosen which has the maximum number of X's which satisfy
the following conditions->
(1) num(R,C,X) is a part of the stable model with R ∈ {1..n} and C ∈ {1..n}
(2) prime(X) is also a part of the stable model implying X is a prime number
```

OUTPUT

14	3	28
29	15	1
2	27	16

Maximum Spaced Magic Square

- The user is allowed to specify the value of ‘n’ i.e. #rows = #columns as well as the value of ‘sum’ i.e. the value of magic constant.
- However, we want to optimize (i.e. find the “best” solution among the set of “good” solutions) on the basis of maximum amount of absolute distance between numbers kept adjacent to each other.

CODE

```
1 {num(R, C, 1..6*6)} 1 :- R=1..n, C=1..n.

R1 = R2 :- num(R1,_,X), num(R2,_,X).
C1 = C2 :- num(_,C1,X), num(_,C2,X).

#const magic=m.

:- #sum{X : num(R,_,X)} != magic, R=1..n.
:- #sum{X : num(_,C,X)} != magic, C=1..n.

:- #sum{X : num(R,R,X)} != magic.
:- #sum{X : num(R,n+1-R,X)} != magic.

#maximize{|X-Y|,|X-Z| : num(R,C,X), num(R,C+1,Y), num(R+1,C,Z), R=1..n, C=1..n}.

#show num/3.
```

```
That stable model is chosen which has the maximum value of |X-Y| and |X-Z| among
the ones that satisfy the following condition->
num(R,C,X), num(R,C+1,Y), num(R+1,C,Z) are a part of the stable model with
R ∈ {1..n} and C ∈ {1..n}
```

2	29	14
27	15	3
16	1	28

OUTPUT

Minimum Negative Magic Square

- The user is allowed to specify the value of ‘n’ i.e. #rows = #columns.
- However, we want to optimize (i.e. find the “best” solution among the set of “good” solutions) on the basis of the minimum negatively valued number (not by magnitude) used to form the magic square.

CODE

```
1 {num(R, C, -N*N..N*N)} 1 :- R=1..n, C=1..n, N=n+1.

R1 = R2 :- num(R1,_,X), num(R2,_,X).
C1 = C2 :- num(_,C1,X), num(_,C2,X).

#const magic=(n**3+n)/2.

:- #sum{X : num(R,_,X)} != magic, R=1..n.
:- #sum{X : num(_,C,X)} != magic, C=1..n.

:- #sum{X : num(R,R,X)} != magic.
:- #sum{X : num(R,n+1-R,X)} != magic.

#minimize{X : num(R,C,X), X<0}.

#show num/3.
```

That stable model is chosen which has the minimum value of X that satisfies the following conditions->
(1) num(R,C,X) is a part of the stable model with R ∈ {1..n} and C ∈ {1..n}
(2) X is less than 0

−5	16	4
14	5	−4
6	−6	15

OUTPUT

Conclusions

- Through this project, we were able to understand the power of Answer Set Programming and how it is an approach to knowledge representation and reasoning.
- By devising an ASP solver for magic squares and it's variations, we got a practical feel of how one can incorporate it for any puzzle provided that they are able to lay down the rules of the puzzle (the constraints) in the language of ASP (i.e. clingo) by using directives like #maximixe and #minimize.
- We were also able to get a good grasp of how one can construct a light python wrapper around the ASP solver for conveniently reading the puzzles from the file and outputting the solutions nicely.

All codes are made available here with proper documentation on how one can use it.