Magic Square ASP Solver

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

—

Swapnoneel Kayal - 200100154
Bibek Saha - 204020012

Outline

1. Problem Statement
2. Work done so far
    a. Pre Midterm Review
    b. Post Midterm Review
3. Few output screenshots
4. Final Deliverables

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.

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

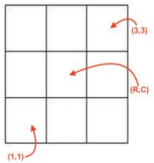
Work done so far

Pre-Midterm Review

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

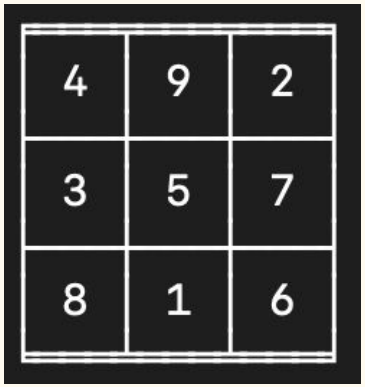
- Every cell of grid is filled with a number between 1 and n^2
- Different squares are filled with different numbers
- Every row sums up to the magic constant
- Every column sums up to the magic constant
- Both diagonals sum up to the magic constant

3. Python wrapper

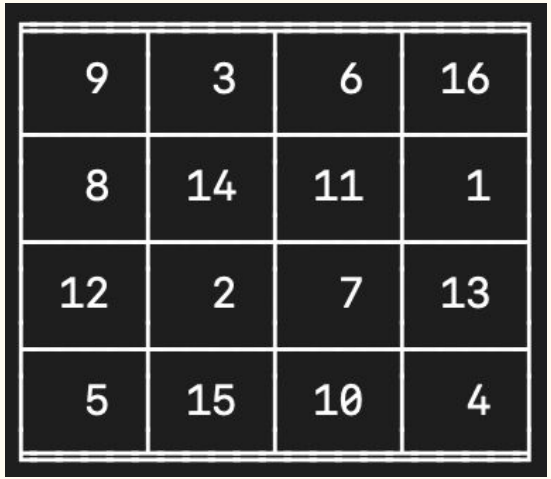
- This was achieved by leveraging the Python subprocess module
- To output the magic square in a readable manner, the tabulate library was used.

Output Screenshots

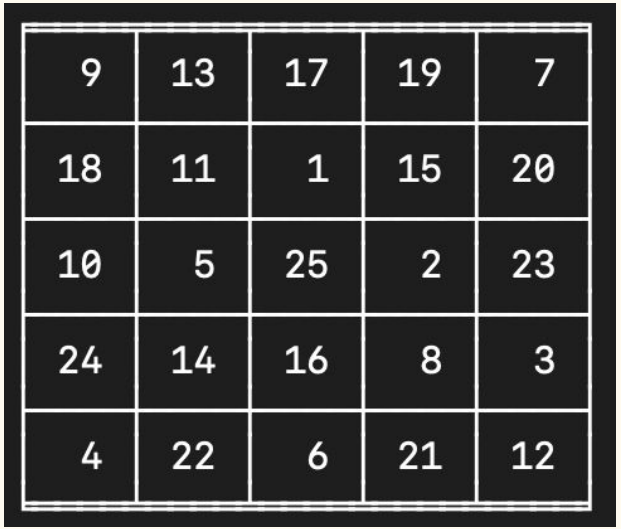
For n = 3:



For n = 4:



For n = 5:



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

Post-Midterm Review

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 Review

(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 Review

(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.

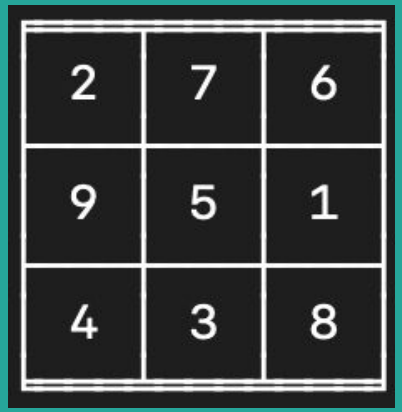
Note that the sum is the user defined magic constant. Appropriate changes were then made to make the sum of rows, columns and the two diagonals equal this sum.

Redefinition of the Tasks

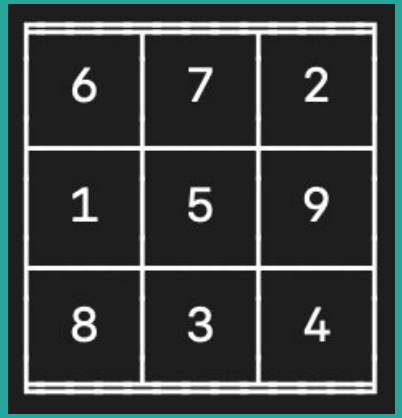
Task 1 - User can specify the number(s) they do not want at a particular cell(s) as well as the magic constant

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

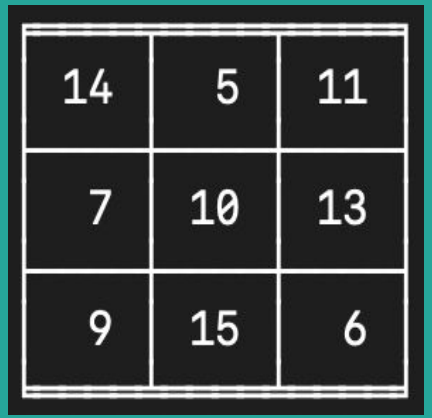
Output Screenshots for Task 1



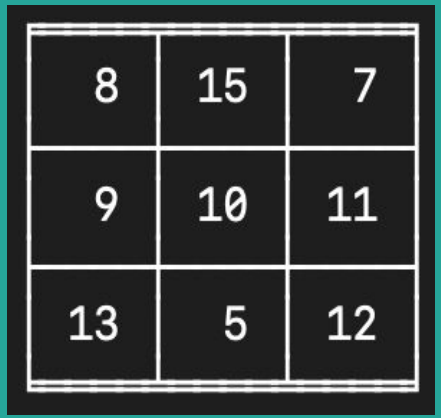
A 3x3 magic square without any constraints



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



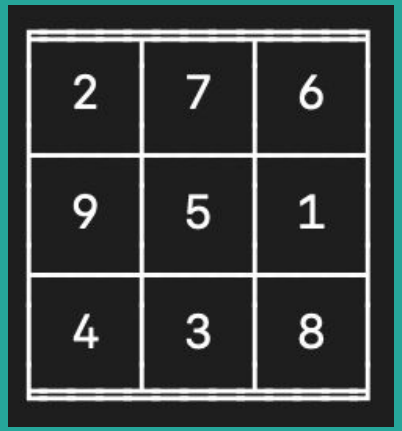
A 3x3 magic square with magic constant = 30



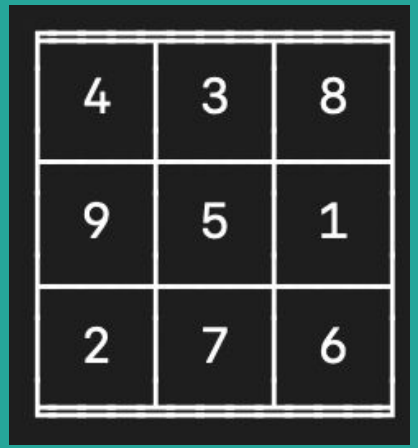
A 3x3 magic square with magic constant = 30 and (1,1,14)

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

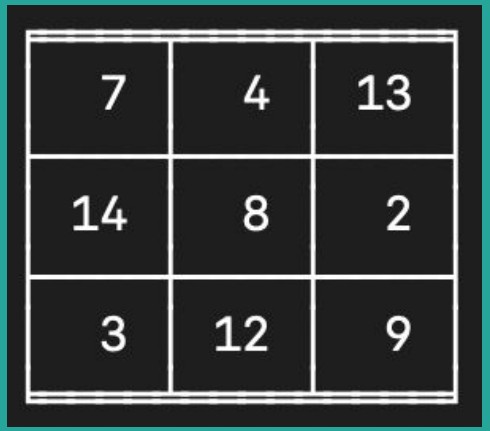
Output Screenshots for Task 2



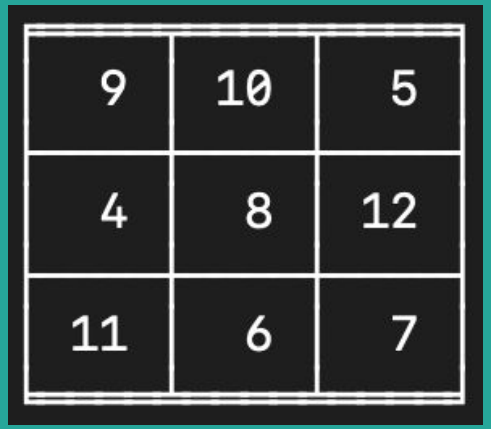
A 3x3 magic square without any constraints



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 deliverables

We feel that we have been able to accomplish all the tasks that were given to us during the mid-term review.

If we look at the project in whole, we can think of incorporating a GUI to improve upon the user-friendliness.

However, we would not be implementing this due to time constraints as well as the very fact that this would take us away from what we are learning in this course i.e. the applications of ASP

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

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)
- We were also able to get a good grasp of how one can construct a light python wrapper around the ASP solver for reading the puzzles from file and outputting the solutions nicely.
- These solvers will either solve the puzzle completely, or give up and produce nothing