ASP Solvers

CS449 Course Project

Project Guide : G. Sivakumar

Swapnoneel Kayal - 200100154

October 12, 2022

Outline

ASP solver

ASP solver

ASP stands for Answer Set Programming. These solvers code the rules for a puzzle in the language of ASP, and use **clingo** to solve them. These puzzles generally have a light python wrapper around them for reading 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, I seek to make an ASP solver for the **magic square** problem and **slitherlink** puzzle

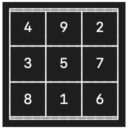
Magic square problem

Magic Square

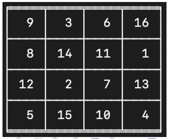
A magic square of order n is an arrangement of n^2 distinct numbers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. A magic square contains the integers from 1 to n^2.

The constant sum in every row, column and diagonal is called the magic constant, M. The magic constant of a magic square has the following value:

$$M = \frac{n * (n^2 + 1)}{2}$$



4	9	2
3	5	7
8	1	6



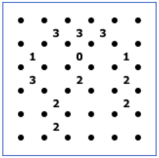
9	3	6	16
8	14	11	1
12	2	7	13
5	15	10	4

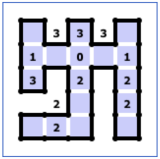
Slitherlink Puzzle

Slitherlink

Slitherlink is played on a rectangular lattice of dots.

- Some of the squares formed by the dots have numbers inside them.
- The objective is to connect adjacent dots horizontally and vertically adjacent dots so that the lines form a single loop that does not cross or branch.
- In addition, the numbers indicate how many edges are adjacent to that number in the solution.

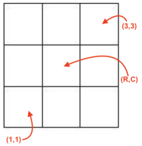




Step 1 : Grid Setup

This step involves encoding the puzzle in a structure that allows us to easily code the rules of any puzzle/problem and lay out the constraints that suits 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 $\in \{1, 2, ..n\}$



With that sorted out, every square of the grid is filled with a number between 1 and n^2.

Step 2 : Puzzle Definition in Clingo

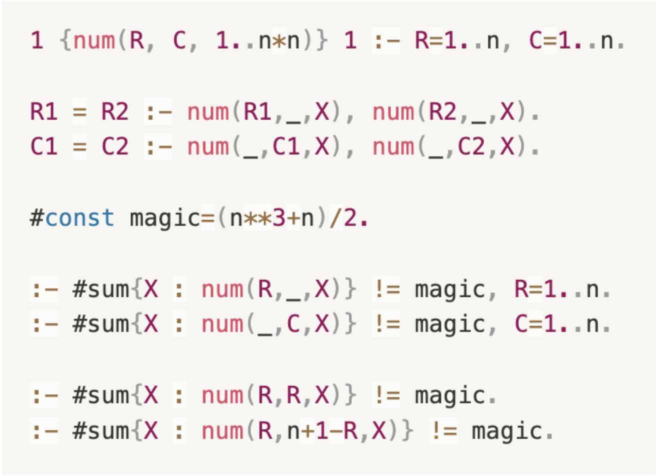
- Every square of grid is filled with a number between 1 & n^2
- Different squares are filled with different numbers
- Every row sums up to the magic constant
- Every column sums up to the magic constant
- Both diagonals sum up to the magic constant

Step 3 : Python Wrapper

This was achieved by leveraging the Python subprocess module for launching a child process which would run the clingo program and direct the output to a file temporarily from where it further gets processed.

In the processing, I have simply used the tabulate library to present the output in a readable manner.

Clingo Solver for Magic Square



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

Python Program for Wrapping

```python
\#! /usr/bin/python
import argparse, subprocess, sys, os
import re
from tabulate import tabulate
import math

parser = argparse.ArgumentParser()

def extract(clingo_path, n=3):
    cmd_clingo = "clingo", "-c", f"n={n}", clingo_path
    f = open('cling-out','w')
    subprocess.call(cmd_clingo, stdout=f)
    f.close()

    file = open('cling-out','r')
    ans = file.readlines()[4]
    temp = re.findall(r'\d+', ans)
    res = list(map(int, temp))

    n = int(math.sqrt(len(res)/3))

    arr=[]
    for r in range(n):
        col = []
        for c in range(n):
            col.append(0)
        arr.append(col)
```

```python
    k=0
    while(k<len(res)):
        row = res[k]
        k+=1
        col = res[k]
        k+=1
        val = res[k]
        k+=1
        arr[row-1][col-1]=val

    print(tabulate(arr, tablefmt='fancy_grid'))

    os.remove('cling-out')

if __name__ == "__main__":
    parser.add_argument("clingo", type=str, default="magic.lp")
    parser.add_argument("--n", type=int, default=3)

    args = parser.parse_args()
    algo = extract(args.clingo, args.n)
```

Demonstration

Work left

Having solved the magic square problem, the next step would be to increase the level of difficulty and try to formulate a slitherlink solver.

The very first step would be to come up with a good data structure and find a way to get around the presence of a global constraint that is generally noticed in loop-style puzzles.

To Do

Some other steps that would be important are:

- Coding the constraint of slitherlink problem in clingo
- Spend some time looking for ways to improve the program i.e. the way clingo searches for the solution
- Code up a light python wrapper around the clingo problem for reading the puzzle from file and outputting the solutions nicely.

References

- **Programming with CLINGO :** https://www.cs.utexas.edu/~vl/teaching/378/pwc.pdf
- **An Introduction to Magic Squares :** https://nrich.maths.org/magic-square-intro
- **Slitherlink :** https://www.conceptispuzzles.com/index.aspx?uri=puzzle/slitherlink/techniques
- **Documentation on subprocess module :** https://docs.python.org/3/library/subprocess.html#using-the-subprocess-module
- **Documentation on tabulate library :** https://pypi.org/project/tabulate/