Pouria KATOUZIAN s192865

Solving Sudokus

INFO9015-1 : Logic for Computer Science

Université de Liège
Faculté des Sciences Appliquées
Année académique 2022-2023

1 Introduction

In this project consists of implementation of a program to create and solve Sudoku puzzles of various sizes through the utilization of the Satisfiability (SAT) problem. This program is developed based on SAT-solving techniques are harnessed to optimize the efficiency of both puzzle creation and solution.

2 Software development

The first step to develop this program is to divide the problem into two main problems which are solving and creating sudoku.

2.1 Solving

In order to solve a Sudoku puzzle, it is imperative to verify the uniqueness of each number within every row, column, block, and square. These verification tasks involve several nested loops. Additionally, it is crucial to confirm that each row, column, block, and square comprises all the numbers from 1 to N exactly once.

2.1.1 SAT

The variable X_{ijk}, is true if the number k is in the cell at row i and column j, and false otherwise.
1. At least one numeral in every case :

$$\bigvee_{k=1}^{N} X_{ijk} \quad \text{for } i,j = 1,\ldots,N$$

2. All columns have numbers 1 to N :

$$\bigwedge_{i=1}^{N}\bigwedge_{k=1}^{N} X_{ijk} \quad \text{for } j = 1,\ldots,N$$

3. All numbers appear at least once in every case row, column, and case :

$$\bigwedge_{i=1}^{N}\bigwedge_{j=1}^{N}\bigwedge_{l=1}^{N}\bigvee_{k=l+1}^{N} (\neg X_{ijk} \vee \neg X_{ijl})$$

$$\bigwedge_{i=1}^{N}\bigwedge_{j=1}^{N}\bigwedge_{k=1}^{N}\bigvee_{l=k+1}^{N} (\neg X_{i+1,j+1,k+1} \vee \neg X_{i+1,j+1,l+1})$$

4. All rows have numbers 1 to N :

$$\bigwedge_{j=1}^{N}\bigwedge_{k=1}^{N} X_{kij} \quad \text{for } i = 1,\ldots,N$$

5. All squares have all numbers :

$$\bigwedge_{i=1}^{N}\bigwedge_{j,k,l,m} X_{(jN+l+1),(kN+m+1),i} \quad \text{for } j,k,l,m = 0,\ldots,n-1$$

6. All squares have at most one occurrence of each number :

$$\bigwedge_{i=1}^{N}\bigwedge_{j,k,l}\bigwedge_{m=1}^{n}\bigwedge_{o=m+1}^{n} (\neg X_{(jN+l+1),(kN+m),i} \vee \neg X_{(jN+l+1),(kN+o),i})$$

7. Uniqueness of values in the Sudoku grid :

$$\bigwedge_{i=1}^{N}\bigwedge_{j,k}\bigwedge_{l=1}^{n}\bigwedge_{m=0}^{n-1}\bigwedge_{o=l+1}^{n}\bigwedge_{p=0}^{n-1} (\neg X_{(jN+l),(kN+m+1),i} \vee \neg X_{(jN+o),(kN+p+1),i})$$

2.1.2 Code

To enhance the performance of the program, unnecessary loops have been avoided, and specific Python method which is **itertools.product()** from import **itertools** have been employed.

```
def sudoku_generic_constraints(myfile, N):
      ... some code ...
    # At least one numeral in every case
    for i in range(1,N+1):
        for j in range(1,N+1):
            for k in range(1,N+1):
                newlit(i,j,k,N)
            newcl()

    for i, j, k in itertools.product(range(1,N+1), repeat=3):
                newlit(i,k,j,N)
    newcl()

    for i, j, k in itertools.product(range(1,N+1), repeat=3):
                for l in range(k+1,N+1):
                    nNewlit(i+1,j+1,k+1,N)
                    nNewlit(i+1,j+1,l+1,N)
                    newcl()
                    nNewlit(i,k,j,N)
                    nNewlit(i,l,j,N)
                    newcl()
                    nNewlit(k,i,j,N)
                    nNewlit(l,i,j,N)
                    newcl()

    for i, j, k in itertools.product(range(1,N+1), repeat=3):
        newlit(k,i,j,N)
    newcl()

    for i in range(1,N+1):
        for j, k, l, m in itertools.product(range(n), repeat=4):
            newlit((j*n) + l + 1, (k*n) + m + 1, i,N)
        newcl()

    for i in range(1,N+1):
        for j, k, l in itertools.product(range(n), repeat=3):
                    for m in range (1,n+1):
                        for o in range(m+1,n+1):
                            nNewlit((j*n) + l + 1, (k*n) + m , i,N)
                            nNewlit((j*n) + l + 1, (k*n) + o , i,N)
                            newcl()
    for i in range(1,N+1):
        for j, k in itertools.product(range(n), repeat=2):
            for l in range (1,n+1):
                for m in range (n):
                    for o in range(l+1,n+1):
                        for p in range(n):
                            nNewlit((j*n) + l , (k*n) + m + 1 ,i ,N)
                            nNewlit((j*n) + o , (k*n) + p + 1 ,i ,N)
                            newcl()
```

2.2 Creating

To generate Sudokus of various sizes, the program first validates the requested size. If the size is valid, the program initializes Sudoku cells and generates a CNF file. It then solves the Sudoku, checks for uniqueness, and modifies the CNF file accordingly. If the Sudoku lacks a unique solution, the program regenerates the puzzle, adds constraints to find a different solution, and repeats this process until a unique solution is achieved. This ensures that the generated Sudoku puzzles adhere to the specified size requirements and possess distinct solutions.

```
1
2  def sudoku_generate(check,size):
3
4      if size not in SIZE:
5          exit("Invalid␣Size␣:␣only␣4,9,16␣and␣25␣are␣allowed")
6      else:
7          n = int(sqrt(size))
8
9      sudoku_cell, nb = initialize_sudoku_cell(size)
10     fill_sudoku_cell(sudoku_cell, nb)
11     sudoku_generate_file(size, sudoku_cell, "generate.cnf")
12
13     while True:
14         sudoku = sudoku_solve("generate.cnf")
15         opf = open("generate.cnf", 'a')
16         sudoku_other_solution_constraint(opf, check, sudoku,
               sudoku_cell)
17         opf.close()
18         sudoku2 = sudoku_solve("generate.cnf")
19         if sudoku2 == []:
20             break
21         sudoku_cell, nb = initialize_sudoku_cell(size)
22         fill_random_cells(check, sudoku_cell, sudoku, size)
23
24     return sudoku_cell
```

3 Feature

The program has successfully fulfilled all the required tasks outlined in the project assignment. It excels in solving Sudoku puzzles, ensuring unicity of numbers in rows, columns, blocks, and squares. Moreover, it adeptly creates Sudoku puzzles with both N numbers and N-1 numbers, showcasing its versatility and robust functionality.

3.1 Test Script

A Bash script has been developed to systematically test all the provided Sudoku puzzles in the "sudoku_student_pack" directory. This script likely automates the process of running the program by applying it to all unsolved Sudoku puzzles provided.

```
1  #!/bin/bash
2
3  for i in {00..03}; do
4
5      xx=$(printf "%02d" $i)
6      python3 sudokub.py -u "sudoku${xx}.txt"
7
8      sleep 1
9  done
```