Autonomous Systems LAB Session 2: Sudoku and SAT Solver

Arnau Colom, Emma Fraxanet

I. MODEL THE PROBLEM AS A PROPOSITIONAL SATISFIABILITY PROBLEM

A. Propositional variables

The propositional variables have to encode the value inside each position in the Sudoku lattice. Based on the properties of the Sudoku lattice, we consider the parameter n, which is the square root in which the structure is based. For example, regular Sudokus have $n = 3$. Therefore we will have $\# = r_{max}c_{max}v_{max}$ variables $X_{r,c,v}$, where $r = 1, .., n^2$ is the row index, $c = 1, .., n^2$ is the column index and $v = 1, .., n^2$ is one of the values that the this position can take.

These propositional variables will be true if the corresponding site in the lattice has the v value. Otherwise they will be false.

B. Clauses for a valid solution in CNF

We will have different clauses that encode the rules of the Sudoku. First of all, we have to make sure each site in the lattice has at least, and at most, one value. Then, we move on to more complex conditions when asking for each row, column and block to contain each value only once. This implies, from the size of the Sudoku, that each value makes one appearance in each row, column and block.

This rules translated to conjunctive normal form can be presented as:

- Each site has at least one value ($(n^2)^2$) clauses).For each site with ($r = r_i, c = c_i$):

$$C_{A,i} = X_{r_i,c_i,1} \vee X_{r_i,c_i,2} \vee ... \vee X_{r_i,c_i,n^2}$$

  Which has to be considered for all sites. We can encode all clauses doing the sum over all row and column values.

$$C_A = \bigwedge_{r_i=1,..,n^2,c_i=1,..,n^2} (X_{r_i,c_i,1} \vee X_{r_i,c_i,2} \vee ... \vee X_{r_i,c_i,n^2})$$

- Each site has at most one value ($(n^2)^2 \frac{n^2(n^2-1)}{2}$ clauses). For each specific site we have to negate the possibility of two values coexisting.

$$C_{A_2,i,j} = (\neg X_{r_i,c_i,v_i} \vee \neg X_{r_i,c_i,v_j})$$

  This has to be considered for all sites and encode all possible pairs of values.

$$C_{A_2} = \bigwedge_{r_i=1,..,n^2,c_i=1,..,n^2,v_{i,j}=1,..,n^2,v_i<v_j} (\neg X_{r_i,c_i,v_i} \vee \neg X_{r_i,c_i,v_j})$$

- Value doesn't repeat inside a row ($(n^2)^2$) clauses). If for a given row and a given value we ask that at least one site has that value, with the expression: $C_{R,i} = X_{r_i,1,v_i} \vee X_{r_i,1,v_i} \vee ... \vee X_{r_i,9,v_i}$, then when we sum over all the values, encoding the sum with *AND* conditions, it is derived that to satisfy the clause there has to be one one different value at each site of that row. This has to be true for all rows, which is encoded in the sum:

$$C_R = \bigwedge_{r_i=1,..,n^2,v_i=1,..,n^2} (X_{r_i,1,v_i} \vee X_{r_i,1,v_i} \vee ... \vee X_{r_i,n^2,v_i})$$

- Value doesn't repeat inside a column ($(n^2)^2$) clauses). Following the same thought as in the previous derivation, we interchange rows and columns to obtain:

$$C_C = \bigwedge_{c_i=1,..,n^2, v_i=1,..,n^2} (X_{1,c_i,v_i} \vee X_{2,c_i,v_i} \vee ... \vee X_{n^2,c_i,v_i})$$

- Value doesn't repeat inside a block ($(n^2)^2$) clauses). This is a little more tricky because we have to define what a block is first. A block will be defined by the parameter n. Note that we defined n^2 at the start because it is needed that the square root is an integer. A new block will start at each column/row value in the set $B_{ind} = 1, n+1, .., n^2-(n-1)$. What we can consider is two variables $i = 0, .., n-1$ and $j = 0, .., n-1$ and, with the fixed row and column indexes from B_{ind} move inside one block or jump from one block to the other. Besides that definition, the condition is similar to the previous ones, we just have to make sure that each value is there for at least one site.

$$C_B = \bigwedge_{r_i,c_i \in B_{ind}, v_i=1,..,n^2} (\bigvee_{s=0,..,n-1, t=0,..,n-1} (X_{r_i+s,c_i+t,v_i}))$$

- Finally , the last set of clauses we need to consider are the initial conditions of the problem, since those are fixed values that have to be true. Being S the set of initial conditions, we need to add:

$$C_S = \bigwedge_{r,c,v \in S} X_{r,c,v}$$

II. IMPLEMENTATION OF THE MODEL

The implementation of the code follows the theoretical framework.

For the problem .....1.4.....2.....5.4.7..8...3....1.9....3..4..2...5.1.....8.6... we obtain the result in Fig.2. Note that we also count the size of the clause list and variables to see if it matches out theoretical expectations, which it does.

III. PROGRAM THAT COUNTS THE NUMBER OF POSSIBLE SOLUTIONS

For this section what we do is create a code that iteratively solves the problem and adds the negated truth assignments of the solution as new clauses, in order to obtain another solution. By doing this iteratively until the problem is not satisfiable anymore we can count the number of solutions.

In Fig. 2 we present the number of solutions obtained for:
a).....1.4.....2.....5.4.7..8...3....1.9....3..4..2...5.1.....8.6... = 1 solution
b).....54.....8.8.19....3....1.6.....34....6817.2.4...6.39.....2.53.2..... = 54 solutions.

If we delete one of the numbers for the initial conditions it takes considerably longer, until the point we had to shut down the program. This performance could be improved by the proposed changes in Section.III.C.

IV. FINAL QUESTIONS

A. What is the size of your SAT problem (number of variables, number of CNF clauses)?

The number of variables will be, if we are considering $n = 3$, $\#var = r_{max}c_{max}v_{max} = (n^2)^3 = 729$. The number of CNF clauses is $\#clauses = |S| + 4(n^2)^2 + (n^2)^2 \frac{n^2(n^2-1)}{2} = |S| + 4n^4 + n^4 \frac{n^2(n^2-1)}{2}$. This

```
arnau@arnau-VirtualBox:~/Desktop/lab2/lab2$ python3 sudoku.py .......1.4.........2...........5.4.7..8...3....1.9....3..4..2...5.1........8.6...
Zero clause 17
First clause 81
Second clause 2899
Third clause 81
Fourth clause 81
Fifth clause 81
Writing SAT problem with 729 vars and 3240 clauses to file "theory.cnf"
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:           729                                         |
|  Number of clauses:            2678                                         |
|  Parse time:                   0.00 s                                       |
|  Eliminated clauses:           0.00 Mb                                      |
|  Simplification time:          0.00 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|       100 |     507     2440     8486 |      894      100      15 | 20.988 % |
|       250 |     507     2440     8486 |      984      250      15 | 20.988 % |
|       475 |     507     2440     8486 |     1082      475      15 | 20.988 % |
|       812 |     507     2440     8486 |     1190      812      15 | 20.988 % |
|      1318 |     507     2440     8486 |     1309     1318      15 | 20.988 % |
|      2077 |     507     2440     8486 |     1440     1340      16 | 20.988 % |
|      3216 |     507     2440     8486 |     1584      839      16 | 20.988 % |
|      4924 |     507     2440     8486 |     1743     1658      15 | 20.989 % |
|      7486 |     506     2432     8460 |     1917     1054      15 | 21.125 % |
|     11330 |     506     2432     8460 |     2109     1739      17 | 21.125 % |
|     17096 |     506     2432     8460 |     2320     1587      16 | 21.125 % |
|     25745 |     505     2424     8440 |     2552     1275      15 | 21.262 % |
|     38719 |     503     2408     8393 |     2807     2334      17 | 21.537 % |
|     58180 |     500     2393     8339 |     3088     2098      14 | 21.948 % |
|     87372 |     497     2363     8250 |     3397     3167      15 | 22.363 % |
|    131161 |     458     2143     7435 |     3737     2870      16 | 27.709 % |
===============================================================================
restarts              : 413
conflicts             : 155289         (158706 /sec)
decisions             : 229766         (0.00 % random) (234822 /sec)
propagations          : 2825688        (2887864 /sec)
conflict literals     : 2513037        (15.69 % deleted)
Memory used           : 23.00 MB
CPU time              : 0.97847 s

SATISFIABLE
Solution: 693784512487512936125963874932651487568247391741398625319475268856129743274836159
Solution in board form:
6 9 3 | 7 8 4 | 5 1 2
4 8 7 | 5 1 2 | 9 3 6
1 2 5 | 9 6 3 | 8 7 4

9 3 2 | 6 5 1 | 4 8 7
5 6 8 | 2 4 7 | 3 9 1
7 4 1 | 3 9 8 | 6 2 5

3 1 9 | 4 7 5 | 2 6 8
8 5 6 | 1 2 9 | 7 4 3
2 7 4 | 8 3 6 | 1 5 9
```

Figure 1: Solution and statistics for Section.II

is easily derivable from the number of sums each clauses encode. For example, clause C_A has sums over r and c and therefore is n^4. On the other hand C_{A_2} has these same two sums but also a sum over possible pairs, which adds the term $\frac{n^2(n^2-1)}{2}$. Note that in our model, n is the square root. The initial conditions of the Sudoku used in Section.II has 17 entries. Therefore, for $n = 3$, the number of clauses is $\#clauses = 17 + 324 + 2916 = 3257$.

B. Dependency of the size of the problem to the size N of the Sudoku board side

From our perspective the Sudoku board size is $N = n^2$. The formula is left as: $\#clauses = |S| + 4N^2 + N^2\frac{N(N-1)}{2}$ and the number of propositional variables as $\#var = N^3$. The size of the SAT problem depends on a polynomial way regarding the board size.

```
arnau@arnau-VirtualBox:~/Desktop/lab2/lab2$ python3 benchmark.py './benchmarks/puzzles1_17_clue' -n=20
Runs: 20, Total time (sec): 16.523, Max: 2.354, Min: 0.334, Avg: 0.826, stdev: 0.413
arnau@arnau-VirtualBox:~/Desktop/lab2/lab2$ python3 benchmark.py './benchmarks/puzzles4_forum_hardest_1905_11+' -n=20
Runs: 20, Total time (sec): 18.885, Max: 1.961, Min: 0.319, Avg: 0.944, stdev: 0.441
```

Figure 2: Results obtained for both of the benchmarks provided

C. Runtime for the two provided benchmarks. What conclusions can you draw from them?

The average runtime per Sudoku is a little bit lower than one second. Even though that is not a long time it could be optimal. One possible way to improve it, as we found in different documentation references, is to use shorter clauses. This apparently improves the performance of *minisat* greatly.

This could be done by separating the n^2 long clauses in C_A, C_R, C_C and C_B into binary clauses. We haven't expanded on that idea for time scope reasons. This could possibly also help with the performance of the counter when deleting one initial value.