Name: Shuqi Liu

October 26th, 2016

Hexadecimal Sudoku Solver using MiniSAT

Frame the Problem

Hexadecimal Sudoku puzzle is a 16x16 Sudoku puzzle filled with hexadecimal numbers from 0 to f. The variable x with a subscript of a three-digit hexadecimal number represents a number at a given grid. In the subscript, the first digit corresponds to the row number, the second digit corresponds to the column number, and the last digit corresponds to the number filled in at the current row and column. For example, ''x_{01a}" represents the grid at row 0 column 1 filled by number 'a' (10 in decimal). There are in total 16^3 = 4096 number of variables from x_{000} to x_{fff}. To use MiniSAT where the variable must be a positive decimal integer, the hexadecimal subscript is converted to the corresponding decimal number plus one.

There are four rules for this game and each of them is written as a combination of some of the variables in a conjunctive normal form (CNF). The first rule is that there must be exactly one number at each grid. We express this rule by asserting that for each grid, there is at least one and at most one number from 0 to f at the grid. The at-least-one rule is expressed by a conjunction of sixteen variables representing the sixteen possible numbers to put at the given grid. For example, at grid (1,2), the at-least-one rule will be a conjunction of x_{120}, x_{121}, ... x_{12e}, and x_{12f}. The at-most-one rule is equivalent to the statement that no two variables at the current grid can be true at the same time. For example, for the grid (1,2), we will assert that "not (x_{120} and x_{121})" which is equivalent to "not x_{120} or not x_{121}". We will then repeat for all two-variable combinations of the sixteen variables at the current grid, which would give $\binom{16}{2}$ = 120 total number of clauses per grid. So, for 16x16 grids, we will have 16*16*(120+1) = 30,976 number of clauses. The process is repeated for the remaining three rules that there is exactly one occurrence of each number 0 to f at each row, column and each 4x4 box. In a discussion with Leyu Fei, we discovered that we do not need to specify the at-least-one rule for the remaining three rules, because in the first rule we have asserted that there must be exactly one number in each grid and now we only need to assert that these numbers on the

board must be all different at each row, column and 4x4 box. Therefore, we have 30976 + $3 * 16 * 16 * \binom{16}{2}$ = 123,136 number of clauses to express the rules of the game.

Next, each puzzle starts with some filled-in numbers on the board. We will simply assert that the variable representing the filled number at any grid has to be true.

Now we pass all the rules in CNF to a SAT solver, MiniSAT in case. If it is satisfiable, there is an assignment of all the variables that would satisfy the rules of the game, which is, by definition, a solution to the Sudoku puzzle. Unsatisfiability means there is no solution to the given puzzle. To find a different solution, we add the assertion that the negation of the first solution is true. If the problem is still satisfiable, we have at least two solutions to the puzzle and the second solution is guaranteed to be different than the first one since it satisfies the negation sentence of the first solution. If the problem is unsatisfiable now, the solution is unique.

Implementation

The Sudoku solver is implemented as a class in Python. The variables are stored in a 16x16x16 matrix using *numpy* where the index is [row, column, number at current position]. To generate the rule for each grid and select all the variables for a grid, we will index by $[i, j, :]$ and repeat for all sixteen rows (i) and sixteen columns (j). Similarly, to generate the uniqueness rule for each row, we will select variables at each row by indexing $[i, :, j]$ and repeat for all sixteen rows and sixteen numbers. To generate the rule for each column, we will select variables at each column by $[:, i, j]$. To generate the rule for each 4x4 box, we will use three nested loops. We will index the rows and columns in a 4x4 group for each number, repeat for 16 numbers per 4x4 box and repeat for the 16 boxes on the board.

The $\binom{16}{2}$ combinations of variables are generated by the *combinations()* method from the *itertools* package. The uniform rules for all puzzles are generated and stored in *ruleUniform1.cnf*. For each puzzle, the rules for filled grids are appended to a copy of the *ruleUniform1.cnf* file. The copy operation is done with the *shutil.copyfile()* method.

Converting the hexadecimal number in the text file to an integer is done with the *int(hex string,16)* method in Python's standard library. Converting the integer variable back to the hexadecimal Sudoku solution is hard coded using the hexadecimal to decimal

conversion formula.

Examples

To find the solution, we will first initialize the *SudukuSolver* class, call the *getUniformRules()* method to generate the uniform rules for all the puzzles, and call the *solve(inputFile)* method with the parameter specifying the name or the relative path of the input file. The solver will print the solution and MiniSAT's performance. The solution will also be written to a text file with the naming convention "prob_#solMapped.txt". The solver will immediately check for a second solution and print out either the second solution or "Solution is unique" if no other solution is found. The new solution will be appended to the same "prob_#solMapped.txt" file.

In the solving process, there will be several intermediate text files generated to process and map the MiniSAT result. These files could be safely ignored. Only the "prob_#.inp" and "prob_#solMapped.txt" files are included in the submission.

An example of the output is attached at the end of the file.

This assignment was my first taste to declarative programming. It is amazing how we could solve the Sudoku puzzle without any search. By simply specifying all the rules using only logic, we are able to find the solution as well as check for uniqueness of the solution in MiniSAT within 1 second (at least for the 10 puzzles given). Imagine if we need to perform depth first search for the Sudoku puzzle, we will have at most 16^{16} = 1.84E19 possible pathes to search through. This assignment certainly showed how powerful the SAT reduction technique could be.

Sample Command Line Output for prob_1.inp

```
File - unknown
/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 "/Users/mac/Desktop
/CSC 339 AI/SudokuSolver/SudukuSolver.py"
============================[ Problem Statistics
 ]=============================
|                                             |
| Number of variables:         4096           |
| Number of clauses:         123136           |
| Parse time:                  0.07 s         |
| Eliminated clauses:          0.01 Mb        |
| Simplification time:         0.12 s         |
|                                             |
============================[ Search Statistics
 ]=============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
==================
|      100 |   699    6915   18059 |    2535      99    15 | 78.809 % |
|      250 |   699    6915   18059 |    2789     249    17 | 78.809 % |
|      475 |   699    6915   18059 |    3067     474    17 | 78.809 % |
|      812 |   699    6915   18059 |    3374     811    19 | 78.809 % |
|     1318 |   697    6915   18059 |    3712    1315    16 | 78.857 % |
|     2077 |   695    6915   18059 |    4083    2072    17 | 78.906 % |
|     3216 |   692    6915   18059 |    4491    3208    20 | 78.979 % |
===============================================================================
==================
restarts              : 19
conflicts             : 3721           (13912 /sec)
decisions             : 7469           (0.00 % random) (27925 /sec)
propagations          : 219140         (819304 /sec)
conflict literals     : 75527          (17.64 % deleted)
Memory used           : 7.65 MB
CPU time              : 0.267471 s

SATISFIABLE

 2870 f4e1 6c9d b35a
 15f9 da7b 3024 86ce
 a6be 023c 5f78 d941
 3c4d 5968 ab1e 2f07

 6e9a 1b50 d38f 4c72
 fb8c 6e23 475a 0d19
 5412 c79d 06eb fa38
 7d03 a84f 29c1 eb65

 9756 2f0a b843 c1ed
 cadb e187 9206 35f4
 8021 36c4 edf5 97ab
 ef34 b5d9 71ac 6820

 b2a8 4316 c5d0 7e9f
 0965 7dbe fa32 148c
 43c7 80f5 1eb9 a2d6
 d1ef 9ca2 8467 50b3
============================[ Problem Statistics
 ]=============================
|                                             |
| Number of variables:         4096           |
```

File - unknown

```
|  Number of clauses:          123137                                  |
|  Parse time:                 0.07 s                                  |
|  Eliminated clauses:         0.01 Mb                                 |
|  Simplification time:        0.18 s                                  |
|                                                                      |
============================[ Search Statistics
 ]============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
=============================================================================
=================
|      100 |   704    7005    18981 |    2568      99    14 | 78.809 % |
|      250 |   704    7005    18981 |    2825     249    13 | 78.809 % |
|      475 |   704    7005    18981 |    3107     474    12 | 78.809 % |
|      812 |   704    7005    18981 |    3418     811    12 | 78.809 % |
|     1318 |   704    7005    18981 |    3760    1317    14 | 78.809 % |
|     2077 |   703    7005    18981 |    4136    2075    15 | 78.833 % |
|     3216 |   701    7005    18981 |    4550    3212    16 | 78.882 % |
|     4924 |   700    6915    18785 |    5005    4911    16 | 78.906 % |
=============================================================================
=================
restarts              : 30
conflicts             : 6208         (16277 /sec)
decisions             : 10157        (0.00 % random) (26630 /sec)
propagations          : 370123       (970417 /sec)
conflict literals     : 97856        (24.05 % deleted)
Memory used           : 8.06 MB
CPU time              : 0.381406 s

UNSATISFIABLE
Solution is unique.

Process finished with exit code 0
```