Project 1: SAT-based Sudoku Solving

Zachary Nasu V00911790
Chloe Blankers V00917921
William McCulloch V00896197
Tristan Nicklaus V00913393

Sud2Sat Implementation Details

This program begins by reading the puzzle that is input through STDIN. It then takes into account the format that it was taken in as by removing the newlines, spaces, and different signifiers for empty values. After this a function is called that generates the clauses for the minimal encoding of the sudoku puzzle. The encodings are as follows (taken from slides) and is followed by order in the code:

1. Each cell contains at least one number
2. Each number appears at most once in every row
3. Each number appears at most once in every column
4. Each number appears at most once in every subgrid

This function calls a function where it gives the encoding based upon the i, j, and digit values of the cell with the formula (`81 * (i - 1) + 9 * (j - 1) + (d - 1) + 1`)

As an aside, there is also code commented out at the end of this function to remove any duplicate clauses which can arise [ie from [(2) and (4)] and [(3) and (4)]]. We were unsure if this was desired for the specifications so we left it commented out.

The function then returns the clauses and the list of already defined cells with values is added to that list of clauses. This new list of clauses is then printed out in cnf format to STDOUT.

Sat2Sud Implementation Details

This program begins by storing the input file from STDIN. The contents of the file will either be a single line (“UNSAT”) if the CNF is not satisfiable or two lines if is satisfiable (“SAT” and the list of positive and negative variables). The contents are stored as an array of strings which is split using the newline character. Next, the program checks if the string at index 0 (first line of the input file), is “UNSAT”. If this evaluates to true, the program terminates as there is no solution. If the statement evaluates to false, the second line of the file is stored as an array split on the space character and excluding the last value (always ‘0’). Next, the list of string values is converted to positive and negative integers and passed to the *solve* function.

In the solve function, we create a new variable *output* and set it to an empty string. After this, we define a *read_cell* function which takes as input *i* and *j* which together represent a single cell of the sudoku puzzle. This function returns a value *d* from 1 to 9 who's inverse encoding (`81 * (i - 1) + 9 * (j - 1) + (d - 1) + 1`) is in the list of assignments. After this function definition, we have nested for loops which go through all of the (i , j) cells in the puzzle. For each combination, the *read_cell* returns a value which is converted to a string and concatenated to the *output*. Once all of the cells have been evaluated, the *output* variable will contain the correctly formatted sudoku puzzle solution which is then written to STDOUT.

Performance testing comments on original task (statistics below)

For the performance testing on the original task, we can see that the number of variables or clauses does not change between the puzzles. There was quite a big variation between the average and worst case for a lot of the metrics (restarts, decisions, conflict literals, propagations, conflicts). Though when it came to cpu time, we didn't see much difference between the average and worst case.

Extended Tasks

Extended task 1

For extended task 1, we reformatted the code so it will take in each puzzle/line and instead of outputting it to stdout, we got it to create a separate cnf file for each of the puzzles. We formatted them in 'top${i}.cnf' where i is the line number.

We also edited the shell script for sud2sat to also call minisat on each of the cnf files. Each puzzle outputs to a file 'assign{i}.txt' file if we desired the answer in sudoku format and the stats all get piped into a file top95test.txt to be later parsed.

Performance

From looking at the performance statistics, we can see that the harder puzzles greatly increased the cpu time, restarts, decisions, conflict literals, propagations, conflicts. For all we see a smaller increase in the average than we do in the worst case.

Extended Task 2

For extended task 2 we added on the first clause given in the project description where 'There is at most one number in each cell' to give the efficient encoding.

Performance

For the performance testing, from the original task, we see a small increase in the amount of almost all the metrics (restarts, decisions, conflict literals, propagations, conflicts) for both the mean and worst case without any visible improvements.

Extended Task 3

For extended task 3 we added on the second, third, and fourth clauses from the project description to create the extended encoding.
The encodings are ‘Every number appears at least once in each row’, ‘Every number appears at least once in each column’ and ‘Every number appears at least once in each subgrid’.

Performance

For the performance testing, what we observe is that by adding the additional encodings we improve the performance of the sat solver in all the metrics (restarts, decisions, conflict literals, propagations, conflicts) over both the original task and extended task 2 with the exception of adding more cpu time.

Testing Results

Original Task (https://projecteuler.net/project/resources/p096_sudoku.txt)

1. Variables:
   Mean: 729
   Worst case: 729
2. Clauses:
   Mean: 8829
   Worst case: 8829
3. Parse time
   Mean: 0.0s
   Worst case: 0.0s
4. Eliminated Clauses
   Mean: 0.0 mb
   Worst case: 0.0 mb
5. Simplification time
   Mean: 0.0s
   Worst case: 0.0s
6. restarts
   Mean: 1.02
   Worst case: 2
7. conflicts
   Mean: 8.08
   Worst case: 101
8. decisions
   Mean: 16.46
   Worst case: 150
9. propagations
   Mean: 935.48
   Worst case: 4237.0

10. Conflict literals
    Mean: 46.64
    Worst case: 709.0
11. Memory used
    Mean: 5.01mb
    Worst case: 5.04mb
12. CPU Time
    Mean: 0.00259594s
    Worst case: 0.003064s

Extended Task 1 (http://magictour.free.fr/top95)

1. Variables:
   Mean: 729
   Worst case: 729
2. Clauses:
   Mean: 8829
   Worst case: 8829
3. Parse time
   Mean: 0.0s
   Worst case: 0.0s
4. Eliminated Clauses
   Mean: 0.0 mb
   Worst case: 0.0 mb
5. Simplification time
   Mean: 0.0s
   Worst case: 0.0s
6. restarts
   Mean: 1.9263157894736842
   Worst case: 10
7. conflicts
   Mean: 152.61052631578949
   Worst case: 1485.0
8. decisions
   Mean: 259.3263157894737
   Worst case: 2301.0
9. propagations
   Mean: 6683.5684210526315
   Worst case: 62805.0
10. Conflict literals
    Mean: 1876.1368421052632
    Worst case: 20678.0
11. Memory used
    Mean: 5.0543157894736845mb
    Worst case: 5.12mb
12. CPU Time
    Mean: 0.003300442105263158s
    Worst case: 0.009888s

Extended Task 2 (https://projecteuler.net/project/resources/p096_sudoku.txt)

1. Variables:
   Mean: 729
   Worst case: 729
2. Clauses:
   Mean: 11745
   Worst case: 11745
3. Parse time
   Mean: 0.0s
   Worst case: 0.0s
4. Eliminated Clauses
   Mean: 0.0 mb
   Worst case: 0.0 mb
5. Simplification time
   Mean: 0.0s
   Worst case: 0.0s
6. restarts
   Mean: 1.04
   Worst case: 2
7. conflicts
   Mean: 8.86
   Worst case: 106
8. decisions
   Mean: 17.56
   Worst case: 148
9. propagations
   Mean: 1056.56
   Worst case: 7092
10. Conflict literals
    Mean: 57.1
    Worst case: 948
11. Memory used
    Mean: 5.1418mb
    Worst case: 5.36mb
12. CPU Time
    Mean: 0.0036862s
    Worst case: 0.004299s

Extended Task 3 (https://projecteuler.net/project/resources/p096_sudoku.txt)

1. Variables:
   Mean: 729
   Worst case: 729

2. Clauses:
   Mean: 11988
   Worst case: 11988
3. Parse time
   Mean: 0.0s
   Worst case: 0.0s
4. Eliminated Clauses
   Mean: 0.0 mb
   Worst case: 0.0 mb
5. Simplification time
   Mean: 0.0s
   Worst case: 0.0s
6. restarts
   Mean: 1.0
   Worst case: 1.0
7. conflicts
   Mean: 0.26
   Worst case: 3
8. decisions
   Mean: 1.5
   Worst case: 6
9. propagations
   Mean: 740.96
   Worst case: 936
10. Conflict literals
    Mean: 0.52
    Worst case: 8.0
11. Memory used
    Mean: 5.1842mb
    Worst case: 5.25mb
12. CPU Time
    Mean: 0.0041956s
    Worst case: 0.004573s