FLAC Project — Using SAT to Solve Sudoku

Although SAT is an NP-complete problem in general, certain kinds of large SAT instances can be solved in a reasonable amount of time. In this project, you will use a SAT solver to solve Sudoku puzzles.

Most existing SAT solvers require the input formula to be in Conjunctive Normal Form (CNF). The most commonly used input format is the DIMACS format, whose structure is as shown at the right:

```
p cnf NumVars NumClauses
Clause_1
Clause_2
⋮
Clause_n
```

The variables are assumed to be numbered from 1 to *NumVars*. The i^{th} variable is represented by the positive integer i; the negation of this variable is represented by the negative integer $-i$. A clause is represented by listing the literals in the clause, separated by spaces, followed by a zero ("0"). For example, the below CNF formula is represented in DIMACS as shown at the right.

```
p cnf 4 5
1 0
2 -3 0
-4 -1 0
-1 -2 3 4 0
-2 4 0
```

$$x_1 \wedge (x_2 \vee \neg x_3) \wedge (\neg x_4 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_2 \vee x_4)$$

For more information about DIMACS, see:

`http://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/satformat.ps`

For this assignment we will use the MiniSAT solver, which was the fastest SAT solver in the SAT Competition in 2005 and 2006. You can download and build MiniSAT on the Andrew machines (`unix.andrew.cmu.edu` via ssh or PuTTY) as follows:

```
wget http://minisat.se/downloads/minisat2-070721.zip
unzip minisat2-070721.zip
cd minisat/core
make
```

You can run MiniSat by the following command:

`~/minisat/core/minisat` *InputFile OutputFile*

If the input formula is unsatisfiable, then MiniSat writes "`UNSAT`" to the output file. If the formula is satisfiable, then it writes "`SAT`" and a satisfying assignment.

Exercise: Download `http://www.cs.cmu.edu/~emc/flac09/sample-cnfs.zip` and run MiniSat on these files.

Sudoku. Sudoku is played on an $n \times n$ board, where n is a perfect square. Some cells are prefilled with numbers from 1 to n; the rest are blank. The board is subdivided into $\sqrt{n} \times \sqrt{n}$ blocks. The goal is to fill each cell with a number in such a way that each number from 1 to n occurs exactly once in each row, each column, and each block. The prefilled squares are such that there is a unique solution.

For this assignment, you will write a program which does the following: (1) read a Sudoku puzzle from an input file, (2) encode the Sudoku puzzle as a CNF formula in DIMACS format, (3) use MiniSat to find a satisfying assignment to the CNF formula, (4) read the satisfying assignment produced by MiniSat to generate the solution to the input Sukodu problem. The input and output formats for Sudoku puzzles will be as follows (where 0 indicates that a square is empty):

Sample Input	Sample Output
0 6 0 1 0 4 0 5 0	9 6 3 1 7 4 2 5 8
0 0 8 3 0 5 6 0 0	1 7 8 3 2 5 6 4 9
2 0 0 0 0 0 0 0 1	2 5 4 6 8 9 7 3 1
8 0 0 4 0 7 0 0 6	8 2 1 4 3 7 5 9 6
0 0 6 0 0 0 3 0 0	4 9 6 8 5 2 3 1 7
7 0 0 9 0 1 0 0 4	7 3 5 9 6 1 8 2 4
5 0 0 0 0 0 0 0 2	5 8 9 7 1 3 4 6 2
0 0 7 2 0 6 9 0 0	3 1 7 2 4 6 9 8 5
0 4 0 5 0 8 0 7 0	6 4 2 5 9 8 1 7 3

Due the limited time you have, we recommend using the naive encoding with n^3 boolean variables and $O(n^4)$ clauses. Let the boolean variable $x_{r,c,d}$ be true iff the number d is in the cell at row r, column c. Encode the following constraints, along with the prefilled cells:
- Exactly one number appears in each cell.
- Each number appears exactly once in each row.
- Each number appears exactly once in each column.
- Each number appears exactly once in each block.

Hint: Encode "exactly one" as "at least one and at most one". See slides for more details.

Your solver will take five command-line arguments, as follows:

`./solver` *InputPuzzle OutputSoln MiniSatExec TempToSat TempFromSat*

where *MiniSatExec* is the filename of the MiniSAT executable (e.g., "`~/minisat/core/minisat`"), *TempToSat* is the name of the temporary DIMACS file that your solver will produce, and *TempFromSat* is the solution that MiniSAT will generate.

We suggest writing your solver in C/C++ (compiled by GCC) or Python 2.5. Other possible languages include Java 1.5 (J2SE 5.0), Ruby 1.8.6, and OCaml 3.08. If you would like to use another language, please email the TAs to see if it would be available on the test machine.

Benchmarks and a sample program skeleton are at `www.cs.cmu.edu/~emc/flac09/bench.zip` and `skel.py`. The output of your program on the benchmark inputs must match the provided benchmark solutions exactly, except for possible minor differences in whitespace (we will compare with `diff -Bbs`).