# CMPE 58S: Homework 3

## Fall 2018 Instructor: Alper Sen

## Due Date: Oct 22, 2018 Demo Date: after class

In this homework you will implement a Boolean satisfiability (SAT) solver that takes a set of variables and constraints in conjunctive normal form (CNF) and returns either a satisfying assignment or determines that no satisfying assignment is possible.

In class we discussed the Davis-Putnam-Logemann-Loveland (DPLL) algorithm, which, after almost 50 years, is still the basis for some of the world's fastest SAT solvers. Although SAT is an NP-complete problem, DPLL is able to efficiently exploit CNF clause structure with heuristic techniques like backtracking and unit propagation to solve problems with thousands of variables. Satisfiability problems are often written in a standard text format called "DIMACS CNF" format which the program that you are about to write will have to be able to parse. The format is defined as follows:

Each line that begins with a 'c' is a comment.

- The first non-comment line must be of the form: `p cnf numberOfVariables numberOfClauses'
- Each of the non-comment lines after the first line defines a clause. These lines are space separated lists of variables with a positive value indicating the variable and a negative value indicating the negation of the variable. The line is always terminated with a space followed by a `0'.

For example, the first few lines of this .cnf file

```
c SAT07-Contest Parameters:  unif  k=3  r=4.2  v=19000  c=79800  seed=49237390
c Uniform  UNKNOWN  KSAT  Instance k=3, nbc=79800, nbv=19000, seed=49237390
p cnf 19000 79800
1987  452  3709  0
-9366  -9214  -1319  0
-17825  8919  4446  0
-16248  11524 -13331  0
4850  -908  -7624  0
```

indicate that this is a 19,000 variable SAT problem with 79,800 clauses. The first clause is $(x_{1987} + x_{452} + x_{3709})$, the second clause is $(\neg x_{9366} + \neg x_{9214} + \neg x_{1319})$, etc.

The course web contains SAT problem examples. The archive also contains a verifier in C that will allow you to check your solutions for each SAT problem.

**a)** Implement DPLL. When solving SAT problems, speed is paramount and you will be graded on program correctness and speed of execution. You should use the heuristics discussed in class to make your implementation as fast as possible. These include: Backtracking, Unit propagation, Pure Literal Elimination.

You should not implement other heuristics such as clause learning or conflict-directed backjumping. Your implementation should be in either Java or C/C++. Note, however, that if you choose Java you will need to find creative ways to speed up your program in order to compete with programs that are written in C/C++. If you are writing your program in C or C++, be sure to compile with level 3 optimization (e.g. gcc -O3 satsolver.c -o satsolver).

**b)** Run your program on each of the following satisfiable SAT problems:

problem1.cnf
problem2.cnf
problem3.cnf
problem4.cnf
problem5.cnf

The solutions to each sat problem should be recorded in separate solution files, each named problem NUMBER.sol. Each line of the .sol format contains a variable name followed by a space and the Boolean assignment. For example:

1 1
2 0
3 0
4 0

**c)** For each of the following SAT problems, use your SAT solver to determine whether each problem is satisfiable or unsatisfiable. You do not have to report the actual satisfying assignments for this part.

problem6.cnf
problem7.cnf
problem8.cnf
problem9.cnf
problem10.cnf
problem11.cnf
problem12.cnf

**d)** Each of the problems in part c was generated from the same random CNF generator, but with different numbers of variables and clauses and different ratios of clauses to variables. Report the amount of time that it takes to solve each SAT problem in part c.

Do these times make sense? Specifically: How does the time it takes to solve each problem increase as the number of variables increases (with the same ratio of variables to clauses)? How does the changing the ratio of variables to clauses change the time it takes to solve these randomly generated problems?

Will these timing results hold for real-world SAT problems? What other factors might influence the time it takes DPLL to solve a SAT problem?

**Note:**
You can also compare your solutions to that of MiniSAT. You can download and install it from http://minisat.se/