CSC 339

Fall 2016

Programming Assignment 3

**Due: Friday, October 7 at the beginning of class (10:20 AM)**

**[Note: This file is still in progress …]**

# Do-It-Yourself SAT Solvers!

**This assignment uses several heuristic search techniques to find possibly optimal truth assignments for variables in Boolean formulas.**

The formulas are in **conjunctive normal form** (ANDs of ORs). The **fitness of an assignment** is the number of clauses (ORs) that the assignment satisfies. If there are *c* clauses, then the highest fitness is bounded by *c*. However, if the formula is not satisfiable, then you cannot simultaneously make all *c* clauses true.

You will use **two** of the following techniques:

* \*the DPLL algorithm;
* \*the WalkSAT algorithm;
* a genetic algorithm;
* hill climbing, also known as local search;
* constraint solving with forward search, and
* resolution.

\*See discussion and pseudocode below.

# To do:

* Select **ALL 32 formulas** in [the data file](http://cs.uky.edu/~teal223/cs463/A3Formulas.zip) A3Formulas.zip (on Moodle).
* Run each of your algorithms on each formula.
* If your algorithm uses randomness, do 10 runs per formula.
* Collect data, for each formula and each run, on (1) the time required and (2) the highest value of c found.
* Put these data in tables or graphs (clearly labeled!)

# To hand in:

* A brief, but clear description of each algorithm and your design decisions in implementing it (e.g., how you set your parameters in the genetic algorithm);
* the data you collected, and
* a brief statement of what you learned.
* Also, upload all code that you wrote for the assignment to Moodle.

# Data files and format info

The file format is a simplified version of the DIMACS format, [as described here](http://www.satcompetition.org/2009/format-benchmarks2009.html). For example, the first two lines of the file 10.40.160707067.cnf (see below) says:

* p cnf 10 40 [This means the formulas are in Conjunctive Normal Form with 10 Boolean variables and 40 clauses.]
* -1 -6 -5 0 [The first clause means (NOT x\_1 OR NOT x\_6 OR NOT x\_5). So the clause is TRUE, for example, if x\_1=0, despite the fact that it could also be the case that x\_6=1. This disjunction of 3 literals is then conjuncted with all of the remaining clauses (lines).
* The 0 terminates the line, but in all the files, I think an EOL follows each 0.

To DEBUG your code, (only) you can start with the 10 (easy) formulas in the file A3\_tests.zip in Moodle. Note that these small instances are too easy for the actual assignment. My results (from MiniSAT) are:

10.40.160707067.cnf SAT -1 -2 -3 -4 -5 6 7 -8 -9 -10 0

[Note: SAT means the CNF formula in the file is satisfiable, and x\_1=0, x\_2=0, x\_3=0, x\_4=0, x\_5=0, x\_6=1, etc., is a satisfying assignment. (There could be more.)]

10.42.1465130262.cnf SAT 1 2 -3 4 5 6 7 8 -9 -10 0

10.42.504071595.cnf SAT -1 2 -3 -4 5 -6 7 -8 -9 -10 0

10.44.1247388329.cnf UNSAT

10.44.1667358355.cnf SAT -1 -2 3 -4 5 -6 -7 -8 -9 -10 0

10.46.183405239.cnf SAT -1 2 -3 -4 -5 -6 -7 -8 -9 10 0

10.46.623142927.cnf SAT -1 -2 3 4 -5 -6 7 8 -9 10 0

10.48.1494607484.cnf SAT -1 2 -3 -4 -5 6 7 -8 -9 -10 0

10.48.640112774.cnf UNSAT

For the actual assignment, download, unzip, and use the formulas in the compressed directory A3Formulas.zip on Moodle. The problem set includes:

* 8 satisfiable formulas with 20 variables
* 8 unsatisfiable formulas with 20 variables
* 8 satisfiable formulas with 40 variables
* 8 unsatisfiable formulas with 40 variables

The file names, and your test runs, will tell you which are which. (Source: Prof. Kautz's page for a similar class assignment. Dr. Kautz, if you are reading this, thank you.)

If you want to see what a good SAT solver can actually do, try out [http://minisat.se/MiniSat.html](http://cs.uky.edu/~teal223/cs463/MiniSAT). Feel free to peek at the source code if you wish, but no copying—and if you borrow any ideas be sure to cite.

Note: There are also #SAT (usually pronounced "sharp sat") solvers, such as [this one called SharpSAT](https://sites.google.com/site/marcthurley/sharpsat). These count the number of solutions. While you are NOT writing a #SAT solver, it is helpful at least to know that these exist, and again (CITE), you may find it helpful to look at how the code was implemented.