ECE 473 Project Homework 7

Final project deadline: your code must be submitted by Friday, February 25 at noon. Late submission will be accepted at a 2% per hour penalty until Saturday noon.
Team selection deadline: February 11 at noon.

In this extended homework, we ask you to explore improving a very basic Boolean satisfiability solver that we provide. You may work in teams, if you select and submit your team according to the directions below by Friday, February 11 at noon. There will be a competition among submitted solvers, run on hidden problems, after the homework deadline.

Provided materials

At this time, we are making available the Python code for a basic solver and for a problem generator. You may begin immediately (once you have set your team) in designing better / faster solvers and you may use any technique you dream up or read about. Your code must be written in Python and not use any special libraries more advanced than numpy (ask if you are not sure, including what functions you need from the requested library). Basic functionality libraries are just fine.

We provide the course "local search" lecture for ideas, there is a rich but confusing research literature available that you can also tap, and the problem is accessible enough that your own ideas may also help.

You can generate problems of any size and timeout with the provided code.Try to solve the problems within the default timeout provided.

The code has a command line interface which can be accessed by typing the command `python3 hw7_submssion.py --help` at a Linux command line. The code can take the problem to solve from a file or can generate the problem from command line parameters giving the number of variables and the timeout in seconds (both have default values). The code can also save the generated problem to a provided file name. Finally, the command-line interface allows you to specify as many different algorithms as you like to run on the problem, in sequence. If you prefer to run the code without the command line interface, see the comment block at the bottom of the provided file.

The code has a verbose mode that you will only want to use for quite small problems.

You are encouraged to add multiple algorithms to the command line interface choices and the code that implements it, so you can easily compare on multiple problems. A good initial goal (and minimal project performance will not be less than this goal) would be to be able

to solve most 100 variable problems in less than 30 seconds. A loftier goal would be to solve some 500 variable problems within a minute.

Caveat: The topic of this project is local search, which cannot demonstrate that a formula is unsatisfiable, since it does not systematically explore all possibilities. There is a related area of SAT solution algorithms that systematically solves the satisfiability question, but generally does not scale to the problems that local search can consider. If you simply do a web search on Boolean satisfiability solvers, you will find a lot of literature on that approach that is unlikely to help you here (key tipoffs would be mentions of the DPLL or CDCL techniques). Systematic SAT solving has become quite important in AI and is an offshoot of Boolean constraint propagation, which we will discuss briefly later in the course.

What you will submit

This project has no written component beyond the requirement that your submitted program `hw7_submission.py` must have comments **at the top of the file** naming any algorithmic technique you believe you are using (including making up a brief meaningful phrase for any innovation you have added that you found important for performance and indicating that it is your invention). Your code will provide a `hw7_submission` function filling in the provided skeleton, that accepts a problem and returns an assignment solving the problem if possible.

Your code must be written entirely by your team members; no outside code can be used for this project. If you read outside code to understand an algorithm, you must put it aside before working with your own code. Also, you are responsible to understand how everything you submit is designed to work.

Evaluation

Top competition performers gain a grade benefit described in the syllabus. There is also a minimum standard of performance to avoid a grade penalty described in the syllabus. This homework is not part of the homework grade in the course. (If you have trouble achieving the minimal level of performance, consult with our TAs and we will give you some additional guidance. Work ahead to allow time for this. Everyone who engages this project in a timely and substantial manner should be able to achieve the minimum standard.)

We have not designed the final competition metric in full yet. It is likely to be total run time on a hidden set of problems, with a timeout that grows with problem size. The penalty for not solving a problem is then the timeout becoming the total run time.

We hope to distribute soon an evaluator program that computes a total time for you on an associated set of problems later in the project period, and you are welcome to exchange information about your performance on this program on Piazza threads. But note that it will not guarantee anything about the final project competition.

Team selection

You must select your team by Friday February 11 at noon (we encourage earlier/immediate team selection so you can start to work together). Anyone not submitting a team selection by that time will be expected to work alone.

Please submit a text file containing your team members Purdue email handles (omit @purdue.edu), one per line, with the team leader on the first line. All team members must submit the same team with the same team leader or the team will be rejected.

For example, our course leadership team (omitting UTAs for brevity here) would be described, as follows:

```
givan
gong123
jkim17
```

We may process these submissions with a script; please follow the prescribed format exactly.

Boolean satisfiability

Boolean formulas take on truth values that depend on the values of the underlying Boolean variables. Many practical problems can be coded as Boolean formulas where making the formula true is the goal of the practical problems. Solving the practical problem involves finding ways to set the underlying variables that make the formula true. A modest industry has developed in designing effective "SAT solvers" that can handle very large formulas built from very large numbers of Boolean variables.

In this project, we explore just the beginning of the techniques that can be created. We provide a problem generator that creates large Boolean formulas that do have solutions (satisfying truth assignments to the underlying variables), and a solver that uses a simple random walk in the space of solutions to solve some such problems. Your challenge is to code any idea you like to improve on this performance.

Here are some relevant definitions:

Variable. We assume a set of Boolean variables (e.g. A) is provided. In the code we provide, the variables are represented by the numbers $1..n$, where n is given by the parameter `num_variables`.

Literal. A Boolean literal is either a Boolean variable (e.g. A) or its negation (e.g. $\neg A$). In the code we provide, a positive literal is a number $1..n$ and negative literal is the negation of such a number, $-1..-n$.

Clause. A Boolean clause is a disjunction ("or", $\vee$) of three different literals, usually given as a set: $\{A, B', C\}$ for $A \vee B' \vee C$. In the program a clause is represented as a list of three literals, e.g. `[-5,3,7]`.

3-SAT formula. A 3-SAT formula is a set of clauses, representing the conjunction ("and", $\wedge$) of the clauses. In the provided program, a 3-SAT formula is represented by a list of clauses.

Assignment. A Boolean assignment is choice of "true" or "false" for each Boolean variable. In the provided program, an assignment is a list or a 1-dimensional `np.array` with indices in `range(num_variables)`, such that the index i stores the value of variable $i+1$. Note that the assignments are 0-referenced but the variable are numbered $1..n$. The values stored in the list are either +1 for truth or -1 for falsehood of the associated variable.

Evaluation. Given an assignment, every literal, clause, and 3-SAT formula are evaluated to either true or false. A formula is true for the assignment if all of its clauses are true. A clause is true for the assignment if at least one of its literals is true. A positive literal $i > 0$ represents variable i being true and is true if the assignment has a +1 at 0-indexed position $i - 1$. A negative literal $j < 0$ represents the variable $-j$ being false and is true if the assignment has a -1 at in-indexed position $j - 1$.

Satisfiability. A 3-SAT formula is satisfiable if there exists some assignment under which it evaluates to true.

A provided problem generator accepts parameters `num_variables` and `num_clauses` and generates a satisfiable 3-SAT formula meeting these parameters using the representation just described. The project task is to take this formula and find a satisfying assignment.

Note that not every 3-SAT formula is satisfiable. Local search is essentially incapable of detecting unsatisfiability, and will just run forever failing to find a satisfying assignment. Systematic backtracking can detect unsatsifiability, but scales exponentially to large problems. We provide code for systematic backtracking for you to evaluate and consider. In this project, our problem generator deliberately generates only satisfiable formulas so local search can assume there is a solution.

Moreover, it has been established that most randomly generated 3-SAT formulas with fewer than 4.2 clauses per variable are easily satisfied (they are under-constrained), whereas most 3-SAT formulas with more than 4.2 clauses per variable are unsatisfiable, over-constrained (when drawn from typical random problem generators). For this reason, our problem generator generates only 3-SAT formulas with approximately 4.2 clauses per variable.