|(S|A|T)|———| SAT Solvers Project |———|(S|A|T)|

| Requirements |

<> A computer running Linux or macOS
<> C++17 or later with its standard libraries
<> G++ compiler
<> The “make” build automation tool
<> A terminal to run the program and read program output from

v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^

| Instructions |

[1] In your terminal, change your current directory to the unzipped folder containing the file “makefile.”

==

[2] Run the command “make all” to compile the program.

==

[3] Now, run “./execs” or “cd execs” to locate the programs.

==

[4] For any of the programs, run this and watch it work!

```
./DPLL
./GSAT
./WalkSAT
```

∗ Please note that these algorithms take a long time to perform, each one taking 2–4 hours to complete. After seeing the output and desiring to close the program, just use the shortcut CTRL+C to terminate the program.

v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^

___________ | The Expected Output; What It Means | ___________

Upon first running DPLL, you will see a series of this output:

```
==============================================================
,_____________________________________
| Formula file: uf20-0156.cnf
| -  -  -  -  -  -  -  -  -  -  -  -  -
| Formula Condition: SAT
| Time Taken: 0.00389933 seconds
`_____________________________________
| Final Model: |
'______________'
-11, 9, -1, -12, 15, -14, 4, -2, -16, -5, -10, 6, 3, -18, -8,
-13, -19, 7, -17, 20,


,_____________________________________
| Formula file: uuf50-010.cnf
| -  -  -  -  -  -  -  -  -  -  -  -  -
| Formula Condition: UNSAT
| Time Taken: 0.396771 seconds
`_____________________________________
==============================================================
```

This output above displays first which formula it is trying to satisfy the clauses to, the result it reached in its attempt ("SAT" for satisfiable or "UNSAT" for unsatisfiable), the time taken to finish running, and if the formula is satisfiable, the model/assignment of literals that satisfies the formula.

Upon running GSAT or WalkSAT, you will find similar output:

```
==============================================================
-------------------{-------------------
| Formula file: uf50-09.cnf
| -  -  -  -  -  -  -  -  -  -  -  -  -
| Formula Condition: UNSAT
| Max SAT Clauses: 215
| Time Taken: 1.01126 seconds
-------------------{-------------------
==============================================================
```

This output, however, contains "Max SAT Clauses," which represents the max number of clauses in the formula that could be satisfied by either algorithm and its assigned parameters.

____
v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^

_________________ | Algorithm Descriptions | _________________

Let us first talk about DPLL, or the Davis–Putnam–Logemann–Loveland algorithm. This algorithm has a couple unique features compared to the other algorithms implemented in this project:

<> Most notably, its determination of each formula’s satisfiability should be completely accurate, meaning that if it determines the formula is satisfiable (SAT), then the formula can be satisfied, and the assignment it provides is a solution. Likewise, if it determines it to be unsatisfiable (UNSAT), then the formula truly cannot be satisfied with any given assignment.

<> Unlike the other two algorithms implemented, this DPLL implementation progressively “simplifies” a formula by deleting clauses that are found to be satisfied by an assignment, leaving the goal of the algorithm to find a satisfactory assignment that leaves the algorithm with no clauses left to satisfy.

<> As a safeguard from following down paths of literal assignments that initially seem promising but later yield an unsatisfiable result, DPLL can backtrack from a current assignment in order to try all possible assignments of literals to find a satisfiable assignment.

<> When the algorithm needs to try adding a literal to an assignment, my implementation uses a heuristic function that detects the most commonly–occurring literal in the current formula and chooses it to be the new assignment in finding satisfiability.

==

Now, let us discuss GSAT and WalkSAT. These two algorithms are related but have crucial differences. In order to understand WalkSAT better, let us first understand the basics of GSAT:

<> GSAT creates a completely randomized assignment of literals for each symbol/variable in the formula.

<> GSAT progressively flips different literals in this randomized assignment to see if the flips help satisfy more clauses.

<> GSAT has a flip limit in order to prevent excessively long runtimes or infinite runtimes if the formula is unsatisfiable. In my implementation, I set this to 1024, as far higher values seemed to make a minor difference to its efficacy while increasing runtime dramatically.

<> If the flip limit is reached or all clauses are found to be satisfied, my implementation returns the maximum number of clauses it was able to satisfy.

==

WalkSAT has similarities to GSAT in utilizing randomization to find a SAT assignment and having a flip limit, but it also has differences worth noting:

<> While WalkSAT also randomly flips literals, it also randomly decides to instead pick the literal assignment that yields the greatest number of satisfied clauses.

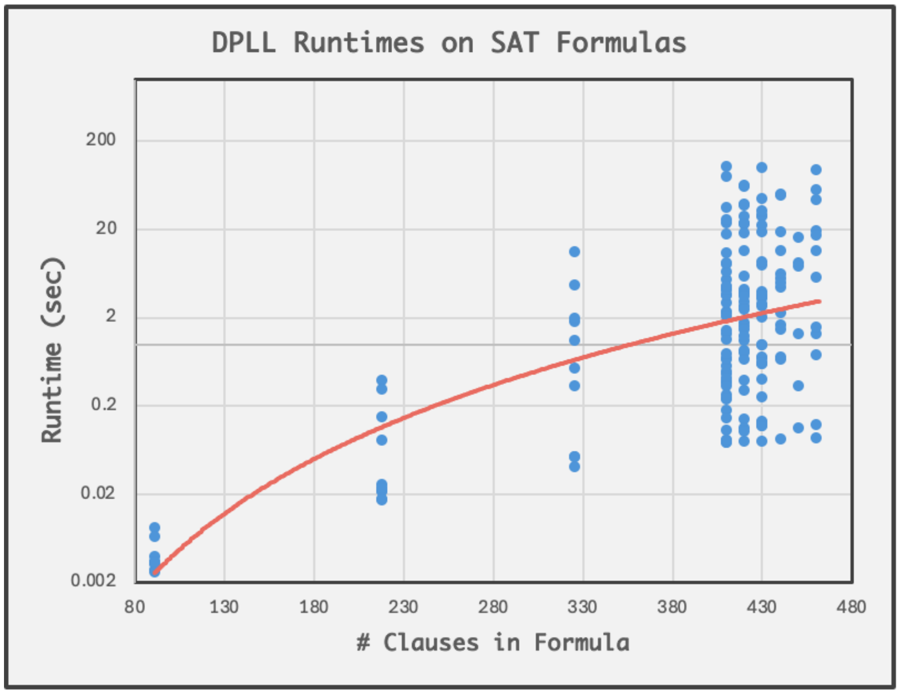
<> This choice between flipping a random literal and flipping a best-choice literal is randomly determined with a probability parameter P. In my implementation, I set parameter P=50 (or 0.5), meaning that there is a 50/50 chance it will pick one or the other.

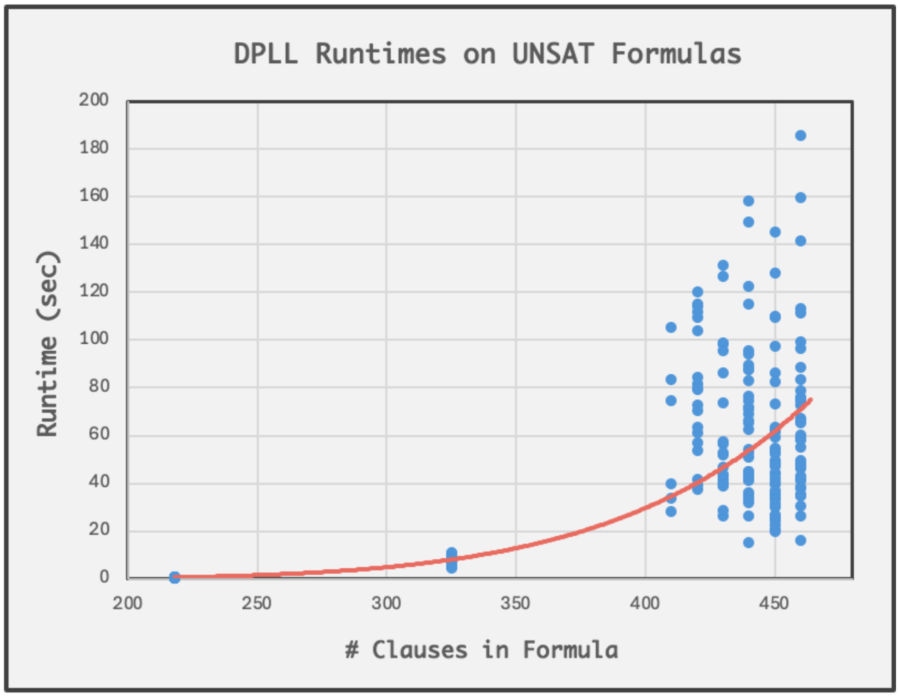
<> My implementation of WalkSAT uses a far higher flip limit (32767) than my GSAT implementation, as the higher value noticeably raised the maximum satisfiable clauses found while maintaining an acceptable runtime.

____
v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^

| Analyzing the Data |

Upon running the programs and extracting important information. The following graphs have been constructed using the data. First, let us look at DPLL's results.

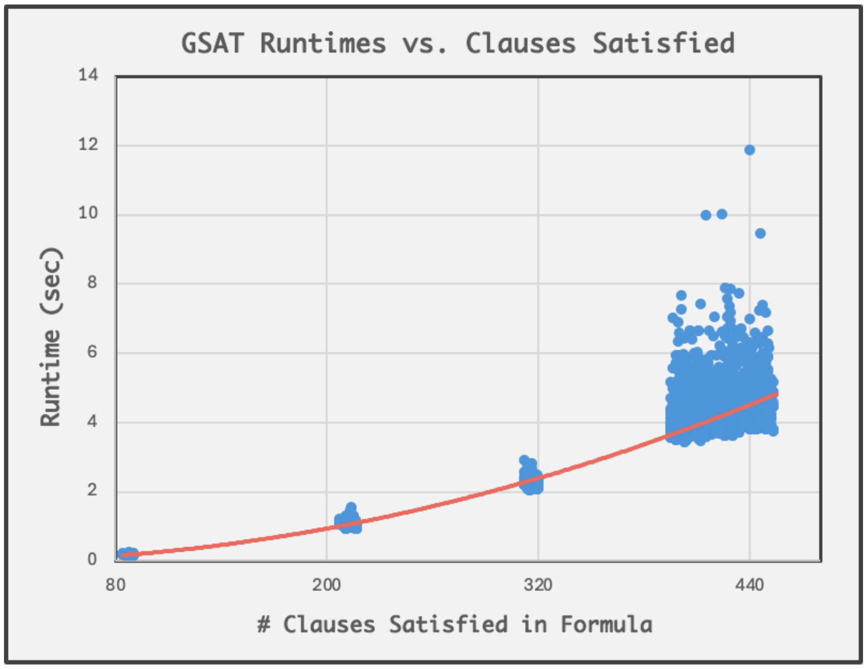


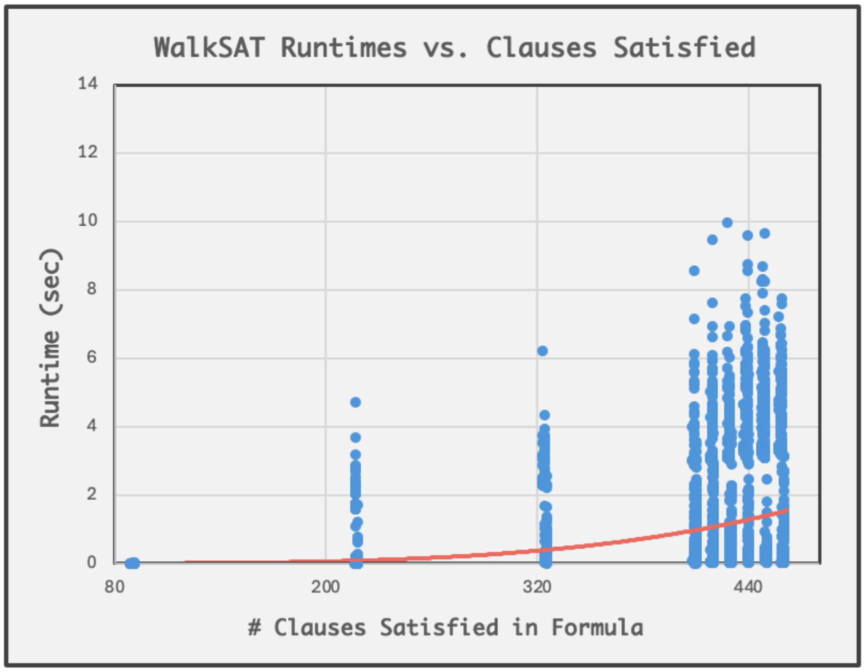


As can be seen above, the runtimes on both satisfiable and unsatisfiable formulas increase with increased clauses (there are also more symbols/variables involved with more clauses in the case of these formulas). However, as can be seen by the logarithmic scale the first graph's vertical axis, it is also clear that the average runtime and runtime variability of DPLL on a satisfiable formula is significantly less than the average runtime on an unsatisfiable formula as the formula size increases.

==

Now, let us take a look at the GSAT and WalkSAT results.





As can be seen above, there are some fairly significant differences in runtimes and maximum satisfied clauses between GSAT AND WalkSAT.

The first and most clear difference between the two is that GSAT has less variation in runtimes compared to WalkSAT, regardless of the number of satisfied clauses.

While GSAT may be fairly consistent in this regard, it is also worth noticing that WalkSAT has a far better runtime on average compared to GSAT, as indicated by the red trend lines. Though not mentioned in these graphs, the sum of each algorithm's runtimes leaves us with 2.11 hours of runtime for WalkSAT and 3.33 hours for GSAT——a substantial difference.

Lastly, WalkSAT has far less variation in the maximum number of satisfied clauses for each formula size (notice the clear-cut lines between pillars of dots in WalkSAT's graph while GSAT has blob-like clusters of dots).

____
v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^

_____________________ | What I Learned | _____________________

After completing this assignment, I learned a lot about the techniques used to try and find solutions, or assignments, to both small-scale and large-scale Boolean formulas. I learned a deeper understanding of the different ways that these algorithmic techniques use to find a satisfiable assignment in as short of a runtime as possible.

Additionally, after experiencing the painful (albeit interesting) runtimes of these algorithms to find or not find satisfiable assignments, I gained deeper understanding of the Boolean satisfiability problem and the N versus NP problem.

____
v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^