# Project TeamworkTemplate

Version 1 9/11/24

A ***separate copy*** of this template should be filled out and submitted by each student, regardless of the number of students on the team. Also change the title of this template to “Project x Teamwork <team> - <netid>”

<table>
<tr><td>1</td><td>Team Name: cander35</td></tr>
<tr><td>2</td><td>Individual name: Cate Anderson</td></tr>
<tr><td>3</td><td>Individual netid: cander35</td></tr>
<tr><td>4</td><td>Other team members names and netids: X</td></tr>
<tr><td>5</td><td>Link to github repository: https://github.com/cateranderson/TOC_Project1.git</td></tr>
<tr><td>6</td><td>Overall project attempted, with sub-projects: Rewrite DumbSAT to use an incremental search through possible solutions</td></tr>
<tr><td>7</td><td>List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary)
<table>
<tr><td>File/folder Name</td><td>File Contents and Use</td></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>DumbSATIncrementer_parseCSV_plotData_cander35.py</td><td>- Parsing CNF Problems (parse_cnf_csv):<br>  - The program reads a CSV input file (or stdin) containing multiple CNF problems.<br>  - It skips comments and identifies the CNF format via the p cnf line, extracting the number of variables (Nvars) and clauses (Nclauses).<br>  - Clauses (sets of literals) are stored in lists, and each problem is appended to the problems list as a tuple: (wff, Nvars, Nclauses).<br>- Checking Clause Satisfaction (is_clause_satisfied):<br>  - For each clause, the program checks if at least one literal satisfies the clause.<br>  - For positive literals, if the corresponding variable is assigned True (1), the clause is satisfied.</td></tr>
</table>
</td></tr>
</table>

| | |
|---|---|
| | - For negative literals, if the corresponding variable is assigned False (0), the clause is satisfied.<br>- Incremental SAT Solver (incremental_sat):<br>  - A recursive depth-first search tries to assign truth values to variables.<br>  - For each variable, it first assigns True (1) and recursively checks if this leads to a satisfying assignment for all clauses.<br>  - If not, it backtracks and tries assigning False (0).<br>  - The base case is reached when all variables are assigned, and it checks if all clauses are satisfied.<br>- Testing a WFF (test_wff):<br>  - For each problem (WFF), the solver tests whether a satisfying assignment exists.<br>  - It records the execution time taken to either find a satisfying assignment or determine the problem is unsatisfiable. |
| Test Files | |
| kSAT_input_cander35.cnf.csv<br><br>Originally Dr. Kogge's kSAT.cnf.csv file, but name edited | - Parsing the CSV properly for useful input (parse_cnf_csv)<br>  - The CSV file contains CNF problems with comment lines starting with `c`, problem definition lines starting with `p` (specifying the number of variables and clauses), and each clause represented by a list of integers (positive for variables, negative for their negations). Clauses end with `0` or may omit it, and empty lines or extra spaces are ignored. |
| Output Files | |
| output_cander35.txt | - Logging Results (log_results) into this .txt file:<br>  - The results for each problem (problem number, number of variables, satisfiability, execution time, and the satisfying assignment if applicable) are written to an output file or standard output. |
| Plots (as needed) | |

| | |
|---|---|
| | <table><tr><td>sat_solver_performance_cander35.png</td><td>- Plotting the Results (plot_data) into this .png file:<br>- After solving the problems, the program reads the output file and plots the number of variables against the execution time.<br>- Satisfiable problems are marked with green circles, and unsatisfiable ones with red triangles.<br>- A line of best fit representing the worst-case complexity (2^n) is also plotted.</td></tr></table> |
| 8 | Individual Student time (in hours) to complete: 20 hours |
| 9 | Your specific activities and responsibilities: I completed the project on my own so all of my submission is done by me |
| 10 | What was personally learned (topic, programming, algorithms): Through this project, I deepened my understanding of the SAT problem and how to approach it using algorithms like incremental search. I learned how to optimize the process by using recursive functions and backtracking, which helped me appreciate the power of depth-first search techniques in solving combinatorial problems. Programming this solution reinforced and strengthened my skills in Python, especially in areas like file handling, recursion, and time-efficient algorithm design. I also became more comfortable working with CSV parsing, managing large datasets, and using libraries like `matplotlib` and `numpy` for visualizing data. Overall, this experience gave me practical insights into problem-solving strategies for NP-complete problems and how to implement these approaches in code. |
| 11 | How team was organized, and what might be improved.: I think what I need to improve is the process of developing my code. I feel that I have a lot of different functionality in just one script, but I think the next time I do a large project like this one, different components should be handled in different scripts for better organization and for debugging purposes. For example a script to parse the input and clean it up, and then a separate script to analyze and test the WFF and produce output, and then another script to plot the output data might have made the process smoother. |
| 12 | Any additional material: n/a |