Project 1 Readme Team Lagunowich

10/18/2024

<table>
<tr><td>1</td><td>Team Name: Lagunowich</td></tr>
<tr><td>2</td><td>Team members names and netids: Luke Lagunowich (llagunow)</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: SAT: Implementing a polynomial time 2-SAT solver</td></tr>
<tr><td>4</td><td>Overall success of the project: Successful. Correctly implements polynomial 2-SAT solver using DPLL algorithm. Highlights success compared to rudimentary SAT solver, DumbSat.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: ~8 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/lukelagunowich/cse-30151-project-1-lagunowich</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). Add more rows as necessary.
<table>
<tr><th>File/folder Name</th><th>File Contents and Use</th></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>code_dpll_lagunowich.py</td><td>Contains polynomial time 2-SAT solver, using DPLL algorithm, and various helper functions.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>check_file_lagunowich.cnf.csv</td><td>Contains 2-SAT problems in CNF form stored in CSV with satisfiability indicated, so we can run 2-SAT DPLL solver and compare outcome of solver to previously known satisfiability outcome.</td></tr>
<tr><td>data_file_lagunowich.cnf.csv</td><td>Contains many of varying size 2-SAT problems in CNF form stored in CSV without satisfiability indicated.</td></tr>
</table>
</td></tr>
</table>

	Used for testing performance (time) of 2-SAT DPLL solver.
Output Files	
output_time _dpll_lagunowich.csv	Contains output from running 2-SAT DPLL solver with data from data file and timing performance. Stores time, number of variables, number of clauses, and satisfiability for plotting purposes.
output_time_dumb_sat_lagunowich.csv	Contains output from running DumbSAT solver with data from data file and timing performance. Stores time, number of variables, number of clauses, and satisfiability for plotting purposes. Used as a baseline to compare performance improvement using a new method that we implemented.
Plots (as needed)	
plots_dpll_lagunowich.png	Contains plot of 2-SAT DPLL solver performance time compared to number of variables in expression. Indicates whether the problem was satisfiable. Indicates trend.
plots_dpll_vs_dumb_sat_lagunowich.png	Contains plot of 2-SAT DPLL solver performance time compared to number of variables in expression as well as plot of DumbSAT solver

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>performance time compared to number of variables in expression. Showcases trends and highlights extreme performance improvement using the new 2-SAT DPLL solver method.</td></tr>
<tr><td colspan="2">Other Files</td></tr>
<tr><td>generate_tests_lagunowich.py</td><td>Generates CSV files for both check and data tests including SAT in CNF form and satisfiability for check files utilizing provided DumbSAT solver to validate.</td></tr>
<tr><td>data_ingestor_lagunowich.py</td><td>Ingests CSV files with SAT in CNF form and returns in form for input into DumbSAT solver and 2-SAT DPLL solver including other metrics like number of variables, number of clauses, and satisfiability.</td></tr>
</table>
* Utilizes code from DumbSAT solver – indicated clearly in code comments.
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries: Python (libraries: csv, random, time)</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
for DPLL 2-SAT solver: list of lists representing clauses in SAT in CNF form</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):
for DPLL 2-SAT solver: the code follows the DPLL algorithm. It tests for unit clauses, then performs unit propagation with the literal in unit clause. It tests for literal purity and then performs pure literal assignment with given literal. Then, it checks its base cases for an empty expression, indicating satisfiability, or for an empty clause, indicating unsatisfiability. Then, it selects a literal from the wff. With which, it attempts to solve the wff recursively by assigning true to the literal or assigning true to the negation of the literal.</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the correctness of your code.
To test the effectiveness of the 2-SAT DPLL solver, I utilized a file containing many SAT</td></tr>
</table>

	problems in CNF form that included their satisfiability, previously determined by DumbSAT, a reliable but slow solver. Then, I used the new 2-SAT DPLL solver to solve each SAT problem in CNF form and compared its output, satisfiable or unsatisfiable, to the previously determined satisfiability. The test passed if for each SAT problem in the file as the new solver correctly determined its satisfiability.
12	How you managed the code development: To effectively manage code, I broke up various functionalities and results of the overall program into different files (ie. code, data, check, plots) with labels. This effectively managed the scope of each file. Additionally, I utilized Git, stored on GitHub, for iterative development.
13	Detailed discussion of results: Overall, the 2-SAT DPLL solver offered extreme improvements to solving 2-SAT problems with many variables in polynomial time compared to DumbSAT, a more rudimentary method. While in the worst cases this new solver still has a 2^n time complexity, in SAT problems with up to 28 variables, the maximum number of variables tested, this new solver performed in polynomial time and significantly outperformed rudimentary methods with exponential time complexity, evidently.
14	How team was organized: I was the only, solo member of my team. However, I managed my workflow by proper scoping and iterative development using Git. This allowed me to manage each working piece of my code most effectively.
15	What you might do differently if you did the project again Next time, I would delve deeper into the worst case performance of the DPLL algorithm used in the 2-SAT solver. One way that I would do this is by optimizing the literal selection step to select the best literal possible to solve the SAT problem the quickest, instead simply popping the literal value from the set of literals. Also, I created my plots using Excel, but next time I would like to automate the plotting process using matplotlib.
16	Any additional material: N/A