Project x Readme Team Svensson-Johnson

Version 1 9/11/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_"teamname"

Also change the title of this template to "Project x Readme Team xxx"

1	Team Name: Svensson-Johnson
2	Team members names and netids Annika Svensson - asvenss2 Brynn Johnson - bjohns33
3	Overall project attempted, with sub-projects: SAT Solving Bruteforce & Backtracking
4	Overall success of the project: Had fun and succeeded in writing the code!
5	Approximately total time (in hours) to complete: 5 hours
6	Link to github repository: https://github.com/bjohns77/Project1-TOC
7	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.

File/folder Name	File Contents and Use
Code Files	
src/ bruteforce_backtrack_SAT_Svensson_Johnson.py	Contains the code for the bruteforcing and backtracking SAT solving problems
Test Files	
input/check_SAT_Svensson-Johnson.cnf	This is the main testing file we used for the code – making sure that out code worked and spit out the solution that was

	expected. This is the cnffile.cnf renamed to fit naming conventions
Output Files	
results/ brute_force_check_SAT_Svensson-Johnson_sat_solver_results.csv	This contains the output for the testing files with the bruteforce code.
results/btracking_check_SAT_Svensson-Johnson_sat_solver_results.csv	This contains the output for the testing files using the backtracking code
Plots (as needed)	
plots/plots_Svensson_Johnson.py	contains the python code to parse the csv file for the output files

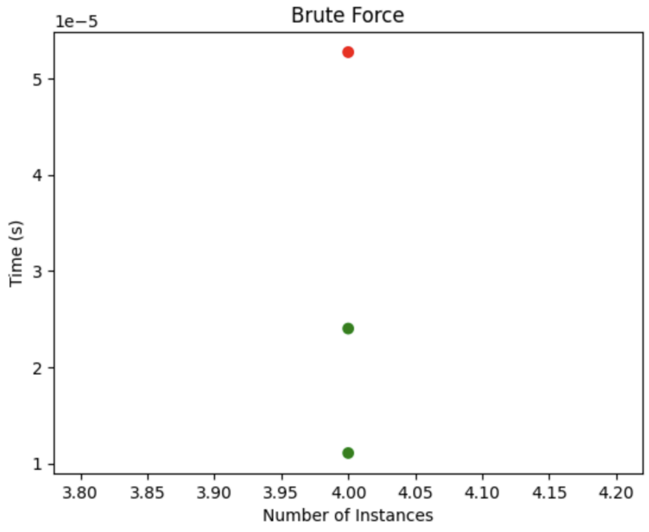
8 Programming languages used, and associated libraries:
Programming language used: Python
Libraries:

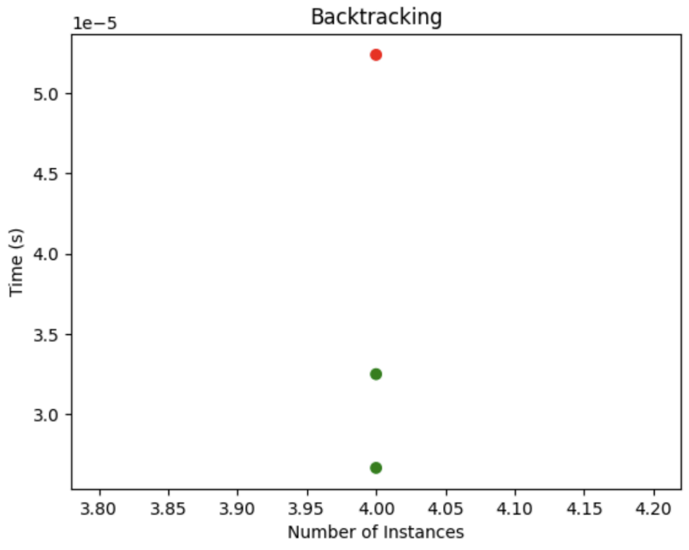
- base code (given):
    - time, csv, json, typing, os, abc, itertools
- added libraries:
    - matplotlib.pyplot as plt (for generating the graph)

9 Key data structures (for each sub-project):

- Bruteforce:
    - each solution is a dictionary with the key as the number and the value as a boolean, true or false
    - broke the input clauses into a list that would be checked against the dictionary
- Backtrack:
    - assignment is a dictionary that stores the current truth assignment of the SAT variables
    - causes is a 2D array that holds each clause, which is a list of signed integers
    - allVars is a list of all variables that occur in the formula, gotten from the clauses
    - unassigned is a list of all the variables not assigned a truth value yet, which is used to assign the next variable
    - variationsTriedStack is a stack that holds all the variations in the form of a tuple (variable, T/F assignment list). The tuple holds the variable and a list of assignments tried

10	General operation of code (for each subproject): - Bruteforce: - Run a while loop until the solution is viable (SAT is satisfiable) or all the solutions have been processed - in this loop the code would: - generate a solution (uses binary generation based on the number of solution it is checking - assigns true or false in the dictionary based on bit in the binary number) - checks if the solution is viable by walking through all the clauses - assumes that the solution is automatically not viable for that clause (good = 0) - if the assignment for that clause sets the clause to true, it continues to loop through the clauses. If it doesn't it breaks out of the loop - if at the end of the for loop good ==1, it is satisfiable - if it is satisfiable, it returns - if it is not satisfiable, it continues to go through the while loop - Backtrack: - Determine whether a CNF is satisfiable by assigning true/false values to variables, one at a time, and backtracking when the current assignment does not distinctly produce a clear answer (satisfiable or unsatisfiable) - The clauseSatisfied function checks if a clause with the current assignments turns the whole CNF true - The clauseUnsatisfied checks if the clause with the current assignments turns the whole CNF unsatisfiable - The final function: - When there is a conflict, like not definitively satisfied or unsatisfied, the stack is used to undo assignments and try different ones - The function decides if it is satisfied with good == len(clauses) - The function decides that is is not satisfied when the backtracking stack empties
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. - test case used: the cnffile.cnf that was given - used because the output was given in the comments so we knew what it was looking for - told us the code was correct - We tried to used the 2SAT and the KSAT tests, but they were taking a long time to run so we did not include them.
12	How you managed the code development We created two different branches that we wrote each of our functions in. We then took turns merging the files into the main branch. After both were merged, we ran the code and took care of any errors.
13	Detailed discussion of results:

	We got the results that we were looking for. The provided code had comments with the desired output for 2 of the test cases in the files. We were able to successfully get these results. We also got graphs that accurately reflected the color and timing of the test cases.
14	How team was organized: Brynn was in charge of coding the bruteforce algorithm and Annika was in charge of coding the backtracking algorithm. We both worked on the report and submission. We also both worked on the plots. We worked well as a team. We did most of it side-by-side which helped because we were able to consult each other when we got confused by the instructions or the file organization.
15	What you might do differently if you did the project again We would like to try all the methods to solve the SAT problem and compare them to each other. It would be cool to see how they all work rather than just knowing how our two work. We would also go back into our code to make it cleaner with better code writing practices, like using list comprehension, etc. We would also like to go back into our code and optimize it.
16	Any additional material: Pictures of the test graphs: 

	Backtracking 1e−5 Time (s) 5.0 4.5 4.0 3.5 3.0 3.80 3.85 3.90 3.95 4.00 4.05 4.10 4.15 4.20 Number of Instances	