CS4244 Project 1

Submission for feedback

by Evelyn Yi-Wen Chen (A0184698E) and Yang Zi Yun (A0184682U)

Instructions to execute:

'mySATSolver.py'
This program will run SAT_Solver on input cnf files (based on input heuristic), then append the result to a text file in 'result/'.

Input:
1. A path to cnf file or a path to directory containing cnf files.
2. Branching heuristic

Output:
1. A **solved.txt** file in **./result/** directory: the results are appended to the text file.

Usage: (3 input parameters required)

```
python3 mySATSolver.py [file/dir path] [heuristic choice: ['random', 'two_clause', 'all_clause', 'max_freq', 'vsids'] or 'all'] [allow debug: 0 or 1]
```

- for example: `python3 mySATSolver.py CS4244_project/sat/uf20-91/uf20-01.cnf two_clause 0`

'experiments.py'
This program will run SAT_Solver in **'mySATSolver.py'** on all cnf files (on all heuristics), then output the results in a csv file.

Input:
1. A path to directory containing cnf files.
2. A path to write the output csv file.

Output:
2. A csv file with experiment results:
   - Each row corresponding to a cnf file, information including:
     - *filename, #variables, #clauses,*
     - *SAT result and time taken (by cryptominisat),*
     - *SAT result, time taken, #branches, #implications (by each heuristic).*
   - A sample file can be found in `result/unsat_uuf50.csv` in this submission.

Usage: (2 input parameters required)

```
python3 experiments.py [cnf_dirname] [output_filename]
```

- for example: `python3 experiments.py CS4244_project/sat/ experiments.csv`

Testing for correctness

- The sat solver is tested on the test set from https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`.
- We tested the solver on the subset of problems in this category:
    - uf100-430 / uuf100-430: 100 variables, 430 clauses - 1000 instances, all sat/unsat
- The output results can be found in this submission in the `results/correct_sat_uf100.txt` and `results/correct_unsat_uuf100.txt`

Heuristics

Heuristics implemented are:

- **random:**
  Decision variable is chosen at random. (Timeout of 10 minutes is set if using random heuristic.)
- **two_clause:**
  Decision variable is chosen from proposition with maximum occurrences in 2-clauses, i.e., with two literals, and break ties randomly.
- **all_clause:**
  Decision variable is chosen from proposition with maximum occurrences in all unresolved clauses, and break ties randomly.
- **max_freq:**
  Decision variable is chosen from proposition with maximum occurrences in all input clauses, unlike all_clause with updates the frequency every time when making decision, this heuristic only calculates the frequency once at the beginning of the program.
- **vsids:**
  Literal with the highest score is chosen.
  Score is maintained for each literal (for both positive and negative propositions), initial score is the number of occurrences of the literal in the input clauses. The score is incremented for all literals in conflict clause. After every 256 conflicts, the score is divided by half to give priority to recently learned clauses.

Current progress:

- The two clause heuristic greatly reduces the solving time required compared to the random heuristics, especially in solving problems with 100 variables.
- However, other heuristics such as all_clause, max_freq and vsids does not show improvements compared to the two_clause heuristics.
- The two_clause heuristic takes 4-7 seconds to solve UNSAT instances of 100 variables.
- Current solver could not handle UNSAT instances of 150 variables efficiently, it will take around 5-10 minutes or more.