CS8803 Logic in Computer Science
Final Report

Mitali Meratwal
mmeratwal3@gatech.edu
GTID: 903925786

1 Helper classes and functions

I have used numpy array to store clauses which vectorises evaluation and assignment.

CNF class: Generate cnf matrix given the clause list processed from text file. Also has a function to generate random cnfs for given n,l,k.

Assignment class: Set assignment to variable given by each dpll step. Convert assignment to a replacement dictionary form for every variable so that for every clause we can evaluate in parallel using numpy arrays.

Evaluation class: Evaluates input cnf under an assignment. If any of the clause is False, the formula evaluates to False. If every clause has at least one variable set to True, we return evaluation as True else we return None and should keep exploring branches in dpll. We also simplify the cnf given current assignment and recurse on the simplified cnf.

2 Heuristic

Unit Preference rule is applied to all implementations, only the splitting rule is different. **Unit Preference rule**: In the DPLL we first get all the clauses with one literal. If there are multiple, we just select the first in the list of such literals (since we always keep selecting unit clauses first before applying splitting rule, the order in this case will not affect performance much). Note if a clause has 3 variables, with 2 set to false, we still consider it unit clause. So unit preference rule generalises to all clauses with one literal unassigned.

Splitting rule: If no unit clause is present, we must select one proposition to proceed.

- **Random**: Randomly select a literal and assign value to proposition such that literal evaluates to True.
- **2-clause**: Select propositions with maximum occurrences in clauses with two literals, and break tie randomly.
- **My heuristic**: Select proposition with maximum occurrence across all clauses and break tie randomly.

3 Numerical results and Plots

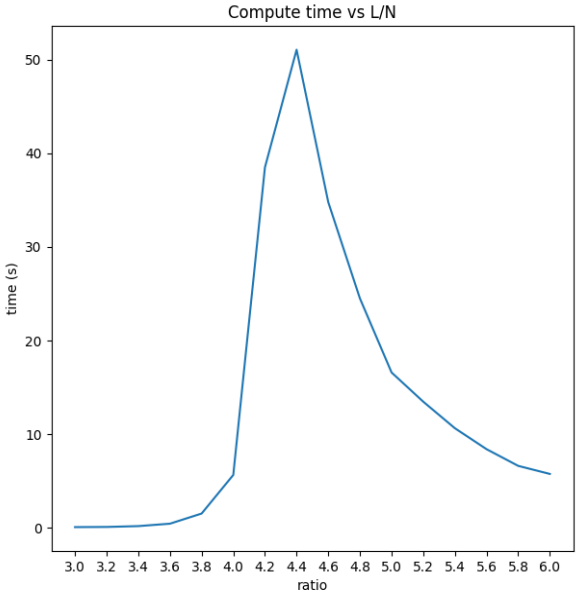

Figure 1: Compute time vs L/N for N=150 on my heuristic

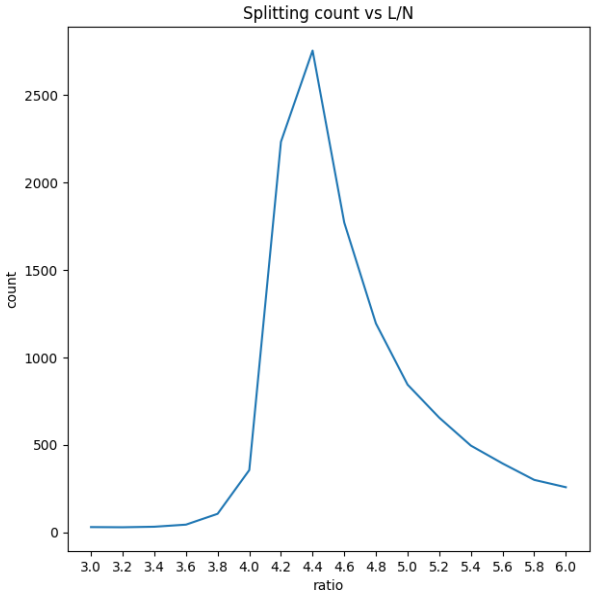

Figure 2: DPLL Split count vs L/N for N=150 on my heuristic

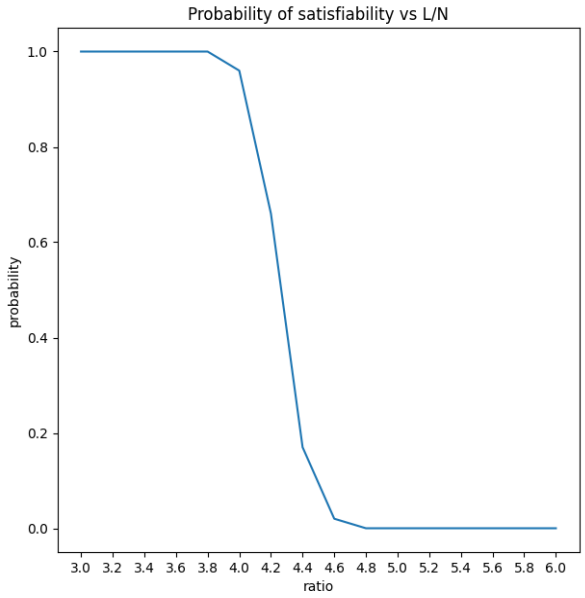

Figure 3: Probability of satisfiability vs L/N for N=150

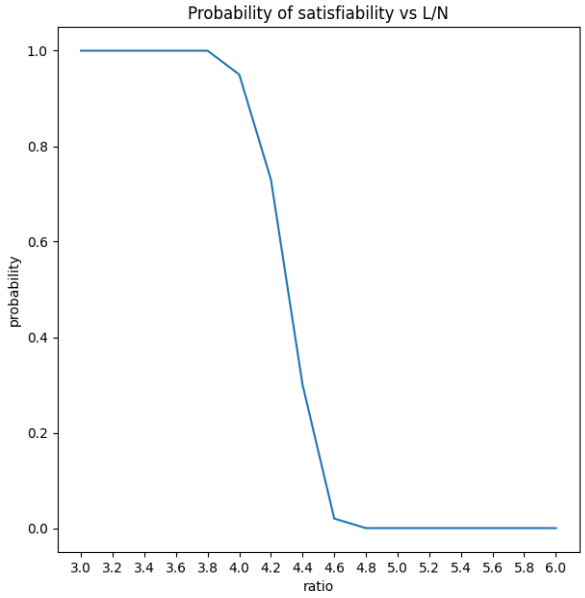

Figure 4: Probability of satisfiability vs L/N for N=175

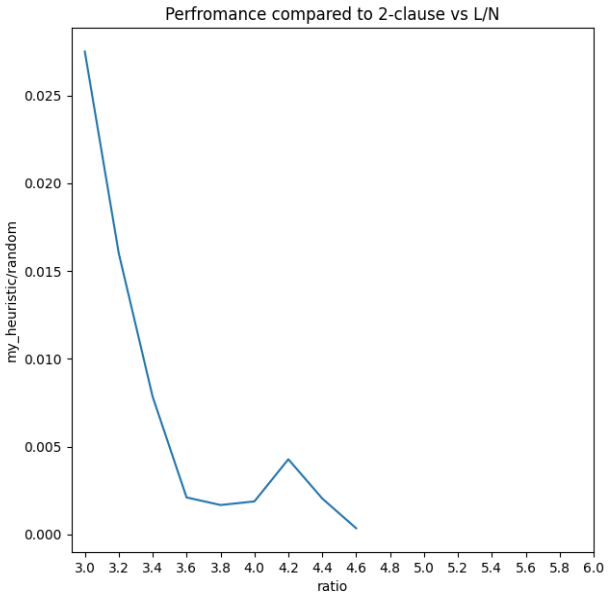

Figure 5: Comparing the performance of my heuristic and random vs L/N for N=150

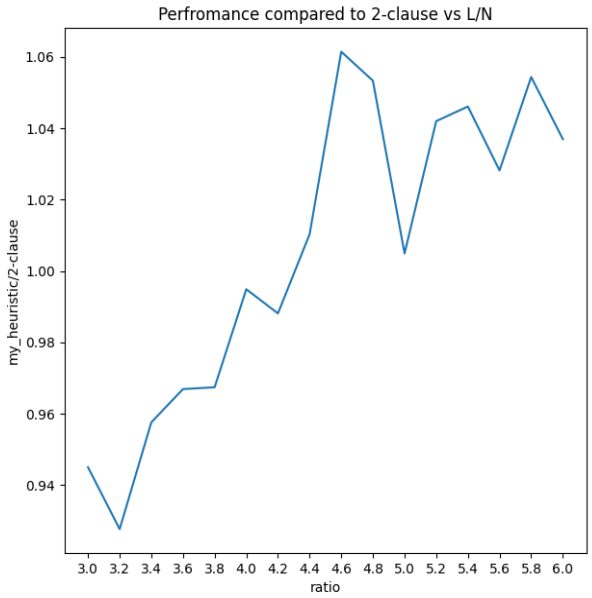

Figure 6: Comparing the performance of my heuristic and two-clause vs L/N for N=150

4 Analysis

From Fig.3 we can see the probability of randomly generated formulas being satisfiable is almost 1 before 4.2, around 50% for 4.3 and almost 0 after 4.2. This suggests that for randomly generated problems and clause to literal variable ratio increases from 3 to 4.2, the number of solutions possible decreases as we are increasing the constraints and hence they become "harder" to solve or the dpll takes non-linear increasingly to find solution (from Fig.1). While after ratio 4.2, are now

over-constrained as we are increasing the number of clauses and hence the problems are unsatisfiable with high probability as the formulas. This results in contradiction being found quickly and so the runtime decreases after 4.2. The probability of satisfiability shifts very little to the left as N increases from 150 to 175. We can guess the shift will move very slowly for larger and larger N. The performance of my heuristic and random heuristic is significantly better than random heuristic as evident from Fig 5 as for random heuristic the time complexity can grow exponentially in the worst case. Note for ratios beyond 4.6 it was taking very long even for one iteration whic made it infeasible to keep running it beyond 4.6. But since random is exponentially increasing and our heuristic is decrasing, we can only expect the curve to go down. By choosing the literal with maximum occurrence first we will reach at a solution or contradiction faster than choosing randomly.
My heuristic is better than 2-clause for ratios upto 4.4 but later 2-clause is almost as same my heuristic.